

APPENDIX C

THE COVENANT TRUST MODEL

SELF-CERTIFICATION OF DIGITAL SIGNATURE KEYS BY CONTRACT

Edwin A. Suominen

A. THE COVENANT TRUST MODEL

I. BACKGROUND

With the Electronic Signatures in Global and National Commerce Act of 2000, the U.S. Congress gave digital signatures the same legal validity as an ink signature on a piece of paper. Now, the sender of an email message, word processing document, or any other type of electronic record that can be construed as a written contract can be legally bound to that record if the recipient can prove that the sender authenticated the record.

Electronic records that are signed with digital signatures can be proven, to a very high level of certainty, to be authenticated by the person who caused the digital signature to be applied to the record. The digital signature can only be applied with a private key, which is an incredibly large number that uniquely corresponds to another incredibly large number, called a public key. The private key, as its name implies, is kept a strict secret by the person who uses it to sign his or her digital signature. Strong cryptographic software ensures that it is "computationally infeasible" (i.e., very difficult, even with very fast computers) to derive the private key from the public key. When a person signs an electronic record with their private key, a digital signature code is produced that anyone can verify against the public key, which is publicly accessible. The slightest change in a document so signed will cause the digital signature to no longer match the document.

The cryptography used in digital signatures is very strong and nearly impossible to tamper with, at least with current technology. But a very old problem remains that technology alone cannot entirely solve. That problem is trust.

The trust problem in digital signatures can be summarized as follows: How do you know that the public key really belongs to the person who says it belongs to him or her? Anyone can create a public key and call it someone else's, then use the corresponding private key to create forged electronic records. The 1998 edition of The Global Trust Register, a printed directory of public keys published by a group of cryptography experts, states the problem as follows: "[T]here is no cheap and effective

way for Internet users to check the validity of public keys on which they may wish to rely."

The experts who wrote The Global Trust Register made that statement in spite of the many efforts by Certification Authorities (CAs) to deploy a "hierarchical trust" model, where trusted third parties check out the identity of persons who own private/public key pairs. A CA such as Verisign, Entrust, or Thawte will add its digital signature to a public key if the public key is tied to the name of a person who physically appears with proper documentation to prove their identity. Recipients of documents signed with the certified public key are then expected to trust that the CA has done its job and that the public key really came from the person whose name is tied to it.

But what happens when one of the many employees at the CA doesn't do his or her job properly? Who is liable for the recipient's reliance on a forged document promising delivery of 10,000 widgets for \$1M when the sender has pocketed the money and run, completely anonymously due to the faceless nature of the Internet? The recipient cannot sue the sender if the recipient doesn't know the sender really was. The recipient's only course of action is to sue the CA for not doing its job. CAs try to avoid liability with disclaimer language in their Certification Practice Statements.

What about tort claims against the CA? Here's what the text Certification Authority Liability Analysis has to say about that:

A CA's liability for tort claims based on negligence **may be limited** by the so-called "economic loss doctrine." The economic loss doctrine provides that claims for purely economic losses based on product defects are not recoverable in tort. The rule holds simply that tort liability does not arise for pure economic loss, but only for personal injury or property damage. The principles behind this rule are that protecting personal injury and property damage claims are more important social policies than pure economic (business) losses, and that economic losses are better protected by negotiated contract allocations rather than through generalized tort law. (Certification Authority Liability Analysis Section 1.1, American Banker's Association, 1998, emphasis added.)

In addition to the problems with "hierarchical trust" that should now be apparent, reliance on the Certification Authority as a trusted third party requires the CA to have an established reputation and to keep its digital house in order for a long time. It doesn't do much good to have a "trusted" third party certifying a digital signature if that third party disappears, loses data, or is found out to have some serious security breach in its infrastructure.

In view of these problems, a system is needed that will translate the direct trust from signer to recipient that self-authenticating ink signatures now provide into the realm of digital signatures. The solution, it turns out, is combining technology with the

trusted authentication that ink signatures and signature witnesses have established over hundreds of years of history.

II. THE COVENANT - AN ANCIENT CONCEPT APPLIED TO TECHNOLOGY

The Covenant Trust Model relies on a person's self-certification of his or her public key and a covenant by that person not to repudiate the public key. The "Covenant of Non-repudiation" legally binds the owner of the public key to any digital signatures created with the corresponding private key. Thus, the liability for proper usage of the private key is placed on the shoulders of the person owning the public key, where it belongs, and legal reliance can be placed upon the public key and any electronic record signed with the corresponding private key.

The covenant is made in an Authentication and Certification Instrument (ACI), a legally signed paper document that contains an identification code positively identifying the public key in question. The document is signed in ink and witnessed by a notary public, thus invoking an authentication system whose trust has been established and is universally recognized by our legal system. An example ACI (see Appendix A-1) contains the following text:

I acknowledge and understand that the consequence of executing this authorization and certification instrument ("Authorization") is that any electronic record accompanied by a digital signature that uniquely corresponds to both the document and the Public Key was signed by me, with a negligible level of doubt. I covenant with any bearer of this Authorization or facsimile copy thereof not to repudiate such digital signature unless I communicate (directly or indirectly) a revocation of the Public Key to the bearer in writing before the signature date.

The ACI includes security features, discussed below, that make it extremely difficult to forge with identification of a different public key, even in a facsimile copy. A person receiving a copy of the ACI (from the signer, from the Internet, wherever) is in possession of a legal instrument that authenticates a public key without the need for trusted third parties. The role of a third party, if one is used at all, is simply to distribute facsimile copies of the ACI. For additional security, the third party can apply its digital signature to the copies of the ACI it distributes to certify them as true copies of the original signed in ink. For example, the third party can authenticate PDF or TIFF files containing facsimile copies of ACIs with a standard SSL (Secure Sockets Layer) certificate issued by a conventional CA.

The conventional "hierarchical trust" model attempts to establish a chain of authenticity to supposedly trusted third parties who are presumed to be doing their jobs properly. In contrast, the covenant trust model establishes a chain of authenticity to a legal covenant, signed with a notarized ink signature on an ACI, in which a public key

owner promises not to repudiate digital signatures corresponding to that public key. The chain of authenticity can begin with initial reliance on the security features of a facsimile copy of the ACI and distribution of the ACI via a trusted web site, email sender, or remote-access viewing software. Higher up on the chain of authenticity, and still convenient to obtain, is digitally-signed certification of the copy by a trusted certifier. Still higher on the authenticity chain is the availability of ink-signed certified copies of the ACI by the original signer or, for a fee, by a trusted certifier. The ultimate link in the chain of authenticity can be provided by making the original notarized, ink-signed ACI paper available for inspection by experts, judges, juries, or attorneys during dispute resolution.

B. IMPLEMENTATION OF COVENANT TRUST VIA THE INTERNET

I. OVERVIEW

A new type of "Certification Authority" will be deployed at **SelfCertify.com** based on the covenant trust model. Selfcertify.com (discussed here in the present tense for convenience) is a certification authority only in the sense that it registers public keys and the identity of persons who claim to own those keys, and certifies that copies of ACIs it distributes are true copies of originals in its possession. It does not certify the identities of the person claiming to own the public keys - those persons make that certification themselves in the ACI.

In addition to registering public keys and distributing ACIs for authentication of those keys, SelfCertify.com can provide standardized digital certificates (e.g., using the X.509 standard) to ensure that its subscriber's public keys can be validated in a manner compatible with conventional public key infrastructure. Again, SelfCertify.com does not pretend that the trust imparted by its digital certificates is based on its confirmation of the identity of its subscribers. Instead, SelfCertify.com makes a policy of only issuing certificates for public keys that subscribers have self-certified with their notarized ink signatures in ACI documents. By signing a public key with its X.509 certificate, SelfCertify.com simply indicates that it has reviewed the original ink ACI and that a copy of the document can be freely downloaded from its Web server.

The use of X.509 or other standard certificates permits SelfCertify.com to live in the world of conventional CAs even though it is based on an entirely different trust model. Users who accept the covenant trust model can install SelfCertify.com's root CA certificate (the "grandfather" certificate that validates all of its individual certificates) into their Web browsers and e-mail applications. As the covenant trust model gains

acceptance in E-commerce, the manufacturers of Netscape Navigator and Internet Explorer can be expected to incorporate SelfCertify.com's root CA certificate into their browsers, alongside the certificates of VeriSign, entrust, and dozens of other CAs. Subscribers who use PGP (Pretty Good Privacy) and are looking for a way to validate their public keys outside PGP's "web of trust" model can submit their public keys to SelfCertify.com for it to be signed by SelfCertify.com's own PGP signature.

Because covenant trust does not require a trusted third party, subscribers' public keys can be validated directly from the subscriber's ACI. The public key of a SelfCertify.com subscriber can be validated by freely downloading a copy of the subscriber's ACI and checking its positive identification of the public key. Thus, no CA certificate is required at all. In fact, subscribers can directly distribute copies of their ACI to anyone who will be relying on signatures corresponding to their public keys.

II. EXAMPLE TRANSACTION USING SELF CERTIFY.COM

Below is a brief description of an example transaction based on covenant trust. In this example transaction, SelfCertify.com serves as a third party for the following:

1. Freely distributing a compact cryptographic software module to signer and recipient with instructions for secure use. The parties use the software for generation of the signer's private/public key pair, generation of the signer's digital signature on an electronic record, and validation of the digital signature against the signer's public key.
2. Accepting credit card payment (with SSL encryption), public key codes, and full legal names of new subscribers to SelfCertify.com.
3. Issuing blank ACIs to new subscribers, upon payment, with instructions for use.
4. Scanning original signed ACIs received from new subscribers and posting digitally certified copies on the web for free downloading.
5. Retaining original ACIs in a vault for inspection by experts, judges, juries, or attorneys during dispute resolution.

For convenience, this example refers to a widget vendor named Alice and a purchaser named Bob. (These names seem to be used in just about every published example of cryptographic transactions.) Alice wishes to sign a purchase agreement acknowledging Bob's payment of \$1M for 10,000 widgets and promises to deliver the widgets immediately. Bob wants to make sure that Alice, the president of Widgets Inc., is the person signing the agreement and not some "man-in-the-middle" imposter.

• Signer Enrollment

Alice visits SelfCertify.com and quickly downloads a copy of "SelfCertify", a simple, compact, secure, and free cryptographic software application for Windows 98/NT/2000, with versions available for various other operating systems. The SelfCertify software installs to the Windows tray as an icon, with various functions selectable by right-clicking on the icon. If she wishes to avoid the need for installation, Alice has the option of simply downloading a single executable file to her desktop and running it from there. For maximum convenience (but possibly less security), a Java version of the software can be offered for execution in a web browser. Because SelfCertify.com serves its pages under SSL with a certificate issued by a conventional CA, Alice is assured that the software is authentic and trustworthy. For additional assurance, Alice reviews statements on the security of the software, written and digitally signed by various cryptographic experts, and validates the signatures of the statements before relying on the software.

Alice then follows the procedures outlined on SelfCertify.com for generating a public key from a secure passphrase. (See Appendix X.) She then gets out her credit card and subscribes to SelfCertify.com with her credit card number, public key code, and full legal name.

Selfcertify.com then issues Alice a custom-generated PDF file, from which Alice obtains two printed pages. The first page is a blank ACI with a space for her driver's license or other photographic ID and the second page is customized security paper with Alice's key code printed repeatedly in the background in an outline font.

Alice tapes her driver's license to the blank ACI in the space provided and places it on the glass of her photocopier, with the security paper at the top of her photocopier's paper supply. She then photocopies the blank ACI to produce an ACI, ready for her signature, with outline digits of her key code throughout its background.

Alice then checks the key code against her public key to make sure it is accurate, goes to the Notary Public down the hall, and executes the ACI in the presence of the notary. The notary examines Alice's driver's license, notes (in the ACI) any security features of it such as a hologram or colored background lines, and signs and stamps the ACI. Alice has now entered into a legally binding covenant with any person bearing the ACI or a facsimile copy of it. (So that she can keep a copy for her files and make certified copies herself, Alice elects to prepare and execute two original copies of the same ACI before the notary.)

Alice mails the executed ACI to SelfCertify.com. Within a few days, SelfCertify.com scans the ACI and posts a copy of it on its web site in PDF or TIFF format. Selfcertify.com stores the original ACI in a vault for possible inspection in the future by experts, judges, juries, or attorneys during dispute resolution. Selfcertify.com then emails Alice the following message:

Your Authorization and Certification Instrument (ACI) has been recorded and you are now listed as a fully enrolled subscriber of **SelfCertify.com** with key ABC01. Once you enter the enrollment password "3f8u2b" in your SelfCertify software, your software will automatically download the latest copy of our public key registry (now including your key) and will automatically validate your digital signatures with the following text in any messages you sign: "The following text has been signed with a public key registered as key ABC01 at SelfCertify.com. Alice B. Costas has signed a written covenant not to repudiate digital signatures created with this public key. To view a copy of this document, click [here](#). The code of this public key is BD7D F2FD EC1C DF14 4811 574F F7CE 7D1E 6EB6 F7E9 CCF7 208B." Persons relying on your digital signature will be able to easily download and inspect a copy of your ACI to legally bind you to that signature.

- **Signer's Digital Signature of Electronic Record**

In her email software, Alice selects the text of her purchase agreement with Bob and right-clicks on the SelfCertify icon in the Windows tray. She then selects the menu item "sign" and, when prompted, enters her private key passphrase. She will probably have to look the passphrase up from a piece of paper in her purse the first few times she uses it. Later, she will put the piece of paper in her safe or destroy it if she trusts her memory enough. If she forgets or loses the passphrase, it's not a big deal. She only needs to create another public key from a new passphrase, cancel her original ACI, and request another one to continue signing records.

As soon as Alice has entered her passphrase, the text she selected in her HTML-formatted email is replaced by text that is identical (including any formatting) except for a block of hexadecimal codes and the following statement in a reduced-size font:

I, Alice B. Costas, have signed this document with my public key, which is registered as key ABC01 at **SelfCertify.com**. To verify this signature, click on <http://SelfCertify.com/validate> to download a compact, virus-free signature verification program that confirms the signature and public key. The software will allow you to obtain a copy of a paper document that you can use to legally bind me to this digital signature. You can also independently validate the public key by clicking on <http://SelfCertify.com/?ABC001> to view a digitally certified copy of the document.

The formatting of the original text is preserved in the signed version. There is no header to the block of signed text because the SelfCertify software automatically calculates the beginning of the signed text block based on the number of signed characters, which is recorded in the signature block. Alice is free to select only a portion

of the text for signature. For example, she may choose not to include letterhead at the top of her letters in the block of text she signs.

Alice can also use S/MIME email software such as Netscape Messenger or Outlook Express to sign email messages using conventional, standardized digital signature technology and the Covenant Trust model, without the need for the SelfCertify.com software. However, she needs to sign an ACI with the SHA1 fingerprint of her S/MIME public key (called a "Digital ID") to authenticate it under the Covenant Trust model. SelfCertify.com then can issue a certificate for her S/MIME public key to authenticate it, based on her ACI.

- **Recipient's Validation of Digital Signature**

Bob receives Alice's digitally signed purchase agreement and downloads the SelfCertify software from the link provided in Alice's signature block. He also downloads a copy of her ACI. Once the software has been installed as an icon, Bob selects Alice's entire e-mail and right-clicks on the icon, then selects "Verify." A window pops up that says:

The following text has been signed with a public key registered as key ABC01 at SelfCertify.com. Alice B. Costas has signed a written covenant not to repudiate digital signatures created with this public key. To view this paper, click [here](#). The code of this public key is BD7D F2FD EC1C DF14 4811 574F F7CE 7D1E 6EB6 F7E9 CCF7 208B.

Since this is a \$1M deal and he has never used the software before, Bob is not content with the software's assertion that Alice has entered into a legally binding covenant not to repudiate her digital signature with this key. Plus, Bob wants to have his lawyer look over the language of the covenant. So he clicks on the "here" link and a viewer window pops up with a TIFF copy of Alice's ACI. He prints out the ACI, notes that Alice's signature (which he recognizes from previous paper-based contracts) has been notarized and that the key code in the ACI is reproduced throughout the background of the document as vertically oriented digits in various outline fonts. The digits intermingle with the signatures, notary stamp, handwritten annotations, and images from Alice's driver's license. The key code digits even show up in the background of Alice's photograph in her driver's license.

Bob needs no further convincing that Alice was the one who signed purchase agreement. His lawyer, however, wants him to check out SelfCertify.com's SSL certificate for the copied ACI. Bob downloads the ACI copy from SelfCertify.com and, with the image of the ACI in his Web browser, clicks on the "security" button of the

browser. The browser provides a certificate issued to SelfCertify.com from a major CA, and Bob's lawyer is satisfied.

If Alice uses S/MIME to digitally signed her message, Bob can simply trust her S/MIME "Digital ID" based on the certificate SelfCertify has issued for it. Thus Alice and Bob can use the Direct Trust model with S/MIME signatures and conventional digital certificates, trusting selfcertify.com as a CA only for inspecting and verifying Alice's CA against the standard covenant language of the ACI, which is published at SelfCertify.com.

Alternatively, Bob can download and review Alice's ACI for her "Digital ID" from the web site of SelfCertify.com. If Bob chooses to download Alice's ACI, he will need to open Alice's "Digital ID," look for her SHA1 fingerprint, and compare it to the fingerprint printed on her ACI. This alternative procedure, while requiring an extra step, provides S/MIME signatures based more directly on the Covenant Trust model, moving closer to the ultimate link in the chain of authenticity, which is the original notarized, ink-signed ACI paper.

C. THE UNDERLYING TECHNOLOGY

The following is a brief listing of various aspects of the inventions discussed in this appendix:

- The key code in the ACI can be printed throughout the background of the entire paper as vertically oriented digits in various outline fonts. The font types, sizes, spacings, and line spacings are varied pseudorandomly in each ACI to make it difficult for an attacker to create an identical field of digits, which the attacker could use to remove the digits (by an XOR operation) from the ACI and substitute his or her own digits. Every bit of text and authenticating indicia in the ACI has background digits running through it. This feature (and possibly other features such as varying the spacing between digits of the text in a coded manner) protects both the signer of the ACI and the person relying on the ACI.
- The ACI can be created with a two-step procedure using a first page that is a blank ACI with a space for her driver's license or other photographic ID and a second page that is customized security paper with the subscriber's public key code printed repeatedly in the background in an outline font. The subscriber tapes his or her photo ID to the blank ACI in the space provided in places it on the glass of her photocopier, with the security paper at the top of the photocopier's paper supply. The blank ACI is then photocopied to produce an

ACI, ready for the subscriber's signature, with outline digits of her key codes throughout its background.

- The ACI can include language that makes it the only printed document of its type that can be accepted as valid. Additional ACIs can be signed electronically for additional keys, but they must be signed with the key that is certified in the original paper ACI. Selfcertify.com attaches digitally signed ACIs (for a fee) to the PDF or TIFF file in which it distributes the original paper ACI. By ensuring that the original printed document disclaims all other documents purporting to bear the singer's handwritten signature, a "strength in numbers" validity system is established that gives the authenticity of a widely distributed ACI, publicly available from a trusted server, far more weight than a single forged copy having a different key code. This feature helps to protect the signer of the ACI.
- The SelfCertify software can employ an ECDSA public key signature system with NIST Elliptic Curve P-192 (equivalent to 80-bit key length of symmetric cipher). The elliptic curve is described by a $GF(p)$ field, where p is prime, to avoid recent attacks on elliptic curves from $GF(2^m)$, where m is a composite of smaller primes. See Smart, N. et al., "Constructive and Destructive Facets of Weil Descent on Elliptic Curves," HP Technical Report HPL-2000-10, 17 January 2000.) A 192-bit public key can be represented by 12 groups of 4 hexadecimal digits. The short key length made possible by elliptic curve cryptography makes it easy for a recipient to visually verify the entire key code against the printed text of an ACI and the background security digits.
- The subscriber can be instructed to use a standardized, pronounceable passphrase made of "pseudowords" with alternating consonants and vowels. The passphrase is designed to be relatively easy to memorize, pronounce, and type and is very secure, with an entropy of about 2^{64} . The passphrase is created with simple, secure system using a piece of paper and a paper clip for random selection of digits.
- A SHA-1 hash of the passphrase can be used as the private key, with the subscriber's full legal name (from the SelfCertify.com directory) incorporated (transparently to the signer) into the passphrase as "salt." The use of salt prevents passphrase attacks using pre-computed hashes of passphrases within the standardized $\sim 2^{64}$ passphrase space.
- Formatting of signed text can be preserved after signing. The added text of the signature block is formatted in an unobtrusive font that does not detract from the appearance of the signed text. The text in the signature block includes a data field

with the number of characters being signed, which avoids the need for a distracting header block (e.g., "-----BEGIN PGP SIGNED MESSAGE-----" in PGP). Documents can also be signed as files, in which case the signature resides in a separate ".SIG" file, as is conventional.

- ACIs can be automatically opened from the software's signature validation window, based on the identification information in the signature block, and displayed or printed from a compact viewing window.

###

2050ED" E462600T

Alice wishes to become a subscriber to SelfCertify.com so that Bob will rely on her public key. However, she doesn't wish to go through the hassle of having a paper document sent to her and having it signed in the presence of a notary. She also wants people to be able to authenticate her public key by hearing a simple recorded statement by her. So, she chooses the "Verbal ACI" option on the SelfCertify.com Web site and enters her phone number and the fingerprint of her public key into the form. The Web site then lists a phone number and an access code and invites her to call the number.

She dials the number (making sure that call blocking is disabled so that SelfCertify.com can detect the phone number she's calling from) and enters the access code using the touchtone keys of her telephone. She then enters into a brief oral exchange with a computer or human operator at SelfCertify.com. The exchange goes something like this:

SC: This telephone call is being recorded for the permanent records of SelfCertify.com, for the purpose of authenticating a public key you are certifying with SelfCertify.com. If you consent to this recording and proceeding with the certification process, please state "I agree" and then recite your full legal name and mailing address.

Alice: I agree. My name is Alice P. Costas, and my address is 537 Main Street, ~~Anytown~~ Arizona 12345.

SC: Now that we have your agreement to record this telephone call and proceed, we will ask that you carefully read the terms of the "Authentication and Certification Instrument." You will be asked to agree to the terms of that document, and your recorded verbal agreement will legally bind you to those terms as if you had signed the document with your ink signature. Please state "Yes, it is" to confirm with the statement entitled "Authentication and Certification Instrument" is now displayed on your web browser at <https://www.selfcertify.com/aci32776> and that the document refers to a public key with fingerprint 2355 7782 1193 8001. You will be given an opportunity to read the document in a minute if you haven't already done so. Right now, we just ask you to confirm that the document is being displayed.

Alice: Yes, it is.

SC: Now we will ask you to ensure that you have read the document. We recommend that you print the document for your records, as you will be bound to its terms if you proceed. Please say "I have read the document" when you have done so.

Alice: Yes, I've read the document.

SC: Now please confirm your legally binding agreement with the terms of the document entitled "Authentication and Certification Instrument," displayed on your web browser at <https://www.selfcertify.com/aci32776> and referring to a public key with fingerprint 2355 7782 1193 8001, on this ____ day of _____, ____ by stating "Yes, I agree to the terms of the document."

Alice: Yes, I agree.

SC: Sorry, you need to state exactly, "Yes, I agree to the terms of the document."

Alice: Yes, I agree to the terms of the document.

SC: Thank you. This includes your verbal certification of your public key. Thank you.

<End of Recording>

SIGNED MEDIA STREAMS

This invention is another aspect of the general concept of calculating a digital signature based on all of the contents of an electronic record except an excluded signature portion. (The general concept advantageously gets around the circular problem of a document essentially signing itself.)

With modern technology, it is difficult to place trust in the authenticity of a video or audio recording. Portions of the recording can be digitally modified in a way that even a careful observer cannot detect. One advantageous aspect of this invention permits a video or audio recording to be validated without requiring special a recording format. Another advantageous aspect of this invention performs frame-by frame authentication of a recording to ensure that the observer is alerted to unauthenticated portions of the recording.

An attached page includes two figures, one depicting the spectrum of an audio recording with an out-of-band transmission of digital signature information, and the other depicting the frame-by frame computation and transmission of digital signatures. During frame T2, a digital signature S1 is computed based on digital samples of the recording within frame T1, and the signature S1 is transmitted (along with other digital samples of the recording) in frame T3. During frame T3, a digital signature S2 is computed based on digital samples of the recording within frame T2, and the signature S2 is transmitted (along with other digital samples of the recording) in frame T4 (not shown). Optionally, digital signature S2 can include (or consist of) an aggregate signature formed from a bitwise modulo sum of the signature and the previous signature.

Advantageously, the video or audio recording can be transmitted and stored independent of any specific digital format, as long as the modulated digital signature information is faithfully conveyed along with the recording information. To allow for degradation of the recording, the signature is preferably computed based on truncated samples. An example of a truncated sample is an audio sample (e.g., noisy 16 bits) that is set to the nearest value within a truncated binary set (e.g., 8 bits). The likelihood of a noisy value being pushed over a boundary between value within the truncated set is small, about $2^{-(x-y)}$ where x is the full set size in bits and y is the truncated set size in bits.

With $x-y=8$, the likelihood is 1 in 255, which represents a fairly significant probability of signature error in a given frame. Consequently, the number of samples in a frame should be kept fairly small.

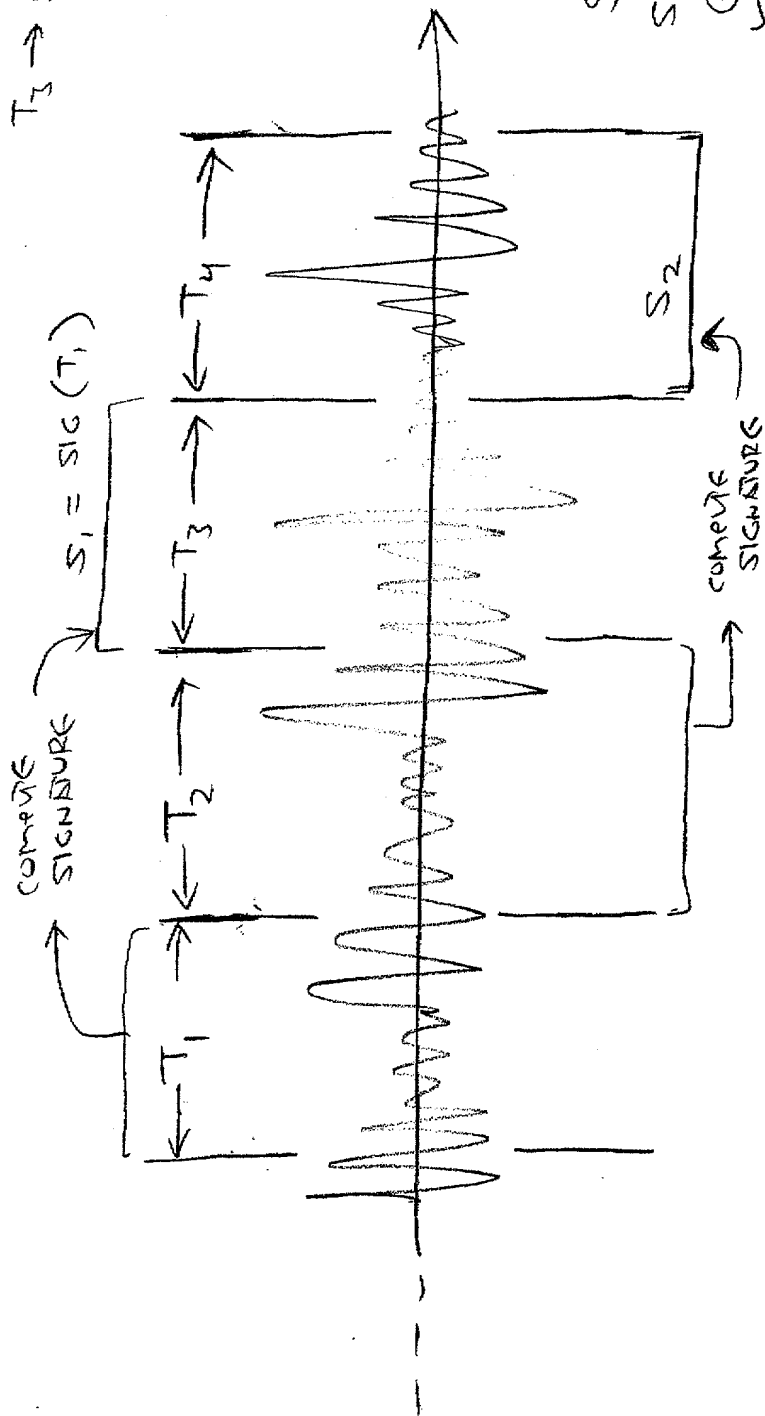
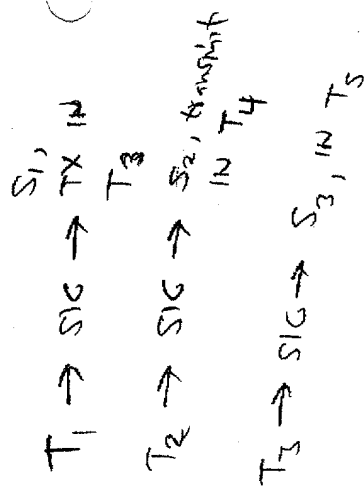
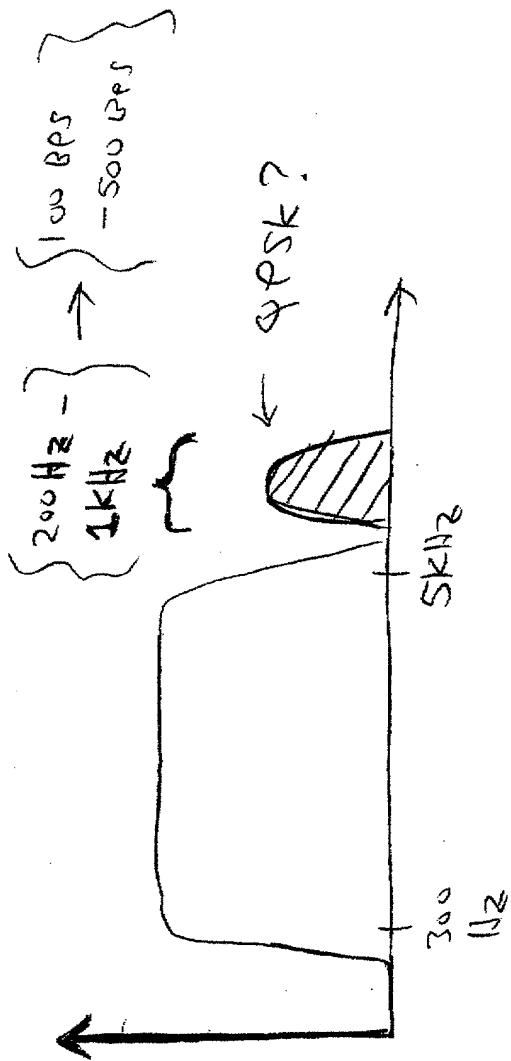
More preferably, speech statistics (i.e., "feature vectors") used in speech recognition (13 spectral magnitude values within 10 ms frames) can be derived from the

recording. If the derivation of the statistics can be made robust in the presence of noise, less signature errors will result.

The media player can authenticate the media stream by including a visually intuitive authentication display. The display can take into account a running statistic of frame authentications, for example by slightly decreasing a "gas gauge" bar for each signature error in a moving average of 32 frames. If frames are short enough, the speech content will not be suspect unless a large gap occurs.

###

2050E0 E462600F



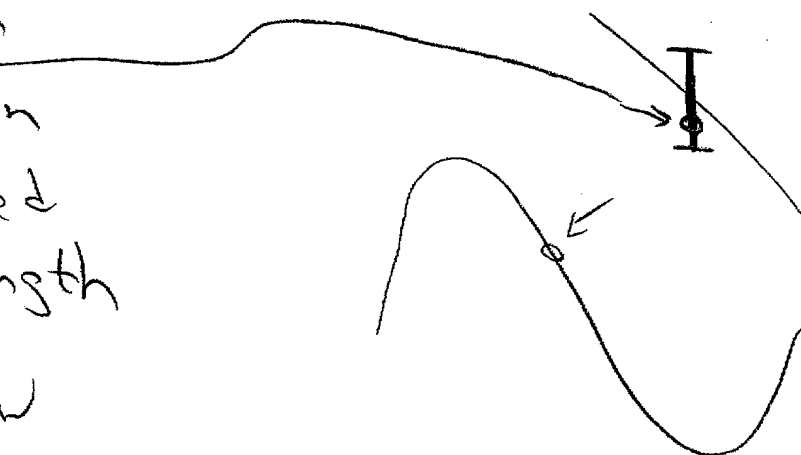
$$S_2 = \text{SIG}(T_2) \oplus S_1 \text{ optional}$$

* SIGNATURE should be compact (ECC)

* Can calculate signature based on truncated sample words should be set to

nearest
value* in
truncated
word length

to allow
for some noise.

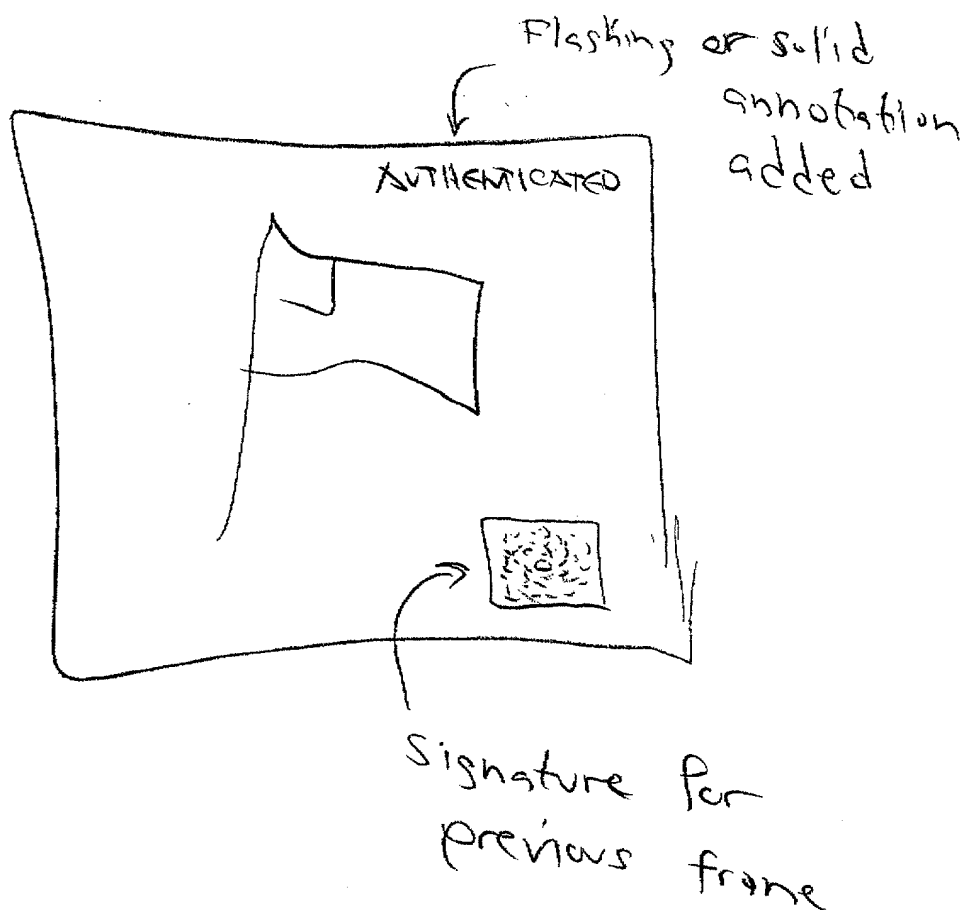


* 256-bit signature (just a guess)

10-second blocks frames

= 25 BPS ← can be subaudible
or DSSS

- * Can put signature barcode (2D) in video image. Will be blanked out in calculation of that block's signature frame



Multimedia Data-Embedding and Watermarking Technologies

MITCHELL D. SWANSON, MEMBER, IEEE, MEI KOBAYASHI, AND
AHMED H. TEWFIK, FELLOW, IEEE

Invited Paper

In this paper, we review recent developments in transparent data embedding and watermarking for audio, image, and video. Data-embedding and watermarking algorithms embed text, binary streams, audio, image, or video in a host audio, image, or video signal. The embedded data are perceptually inaudible or invisible to maintain the quality of the source data. The embedded data can add features to the host multimedia signal, e.g., multilingual soundtracks in a movie, or provide copyright protection. We discuss the reliability of data-embedding procedures and their ability to deliver new services such as viewing a movie in a given rated version from a single multicast stream. We also discuss the issues and problems associated with copy and copyright protections and assess the viability of current watermarking algorithms as a means for protecting copyrighted data.

Keywords—Copyright protection, data embedding, steganography, watermarking.

I. INTRODUCTION

The past few years have seen an explosion in the use of digital media. Industry is making significant investments to deliver digital audio, image, and video information to consumers and customers. A new infrastructure of digital audio, image, and video recorders and players, on-line services, and electronic commerce is rapidly being deployed. At the same time, major corporations are converting their audio, image, and video archives to an electronic form.

Manuscript received July 15, 1997; revised January 15, 1998. The Guest Editor coordinating the review of this paper and approving it for publication was A. M. Tekalp. This work was supported in part by the Air Force Office of Scientific Research under Grant AF/F49620-94-1-0461 and in part by the Advanced Research Project Agency under Grant AF/F49620-93-1-0558.

M. D. Swanson is with Cognicity, Inc., Minneapolis, MN 55344 USA (e-mail: swanson@cognicity.com).

M. Kobayashi is with the Graduate School of Mathematical Sciences, University of Tokyo and IBM Tokyo Research Laboratory, Yamato-shi, Kanagawa-ken 242 Japan (e-mail: mei@trl.ibm.co.jp).

A. H. Tewfik is with Cognicity, Inc., Minneapolis, MN 55344 USA (e-mail: tewfik@cognicity.com) and the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: tewfik@ece.umn.edu).

Publisher Item Identifier S 0018-9219(98)03519-1.

Digital media offer several distinct advantages over analog media: the quality of digital audio, image, and video signals is higher than that of their analog counterparts. Editing is easy because one can access the exact discrete locations that should be changed. Copying is simple with no loss of fidelity. A copy of a digital media is identical to the original. Digital audio, image, and videos are easily transmitted over networked information systems.

These advantages have opened up many new possibilities. In particular, it is possible to hide data (information) within digital audio, image, and video files. The information is hidden in the sense that it is perceptually and statistically undetectable. With many schemes, the hidden information can still be recovered if the host signal is compressed, edited, or converted from digital to analog format and back.

As we shall see in Section II, pure analog data-hiding techniques had been developed in the past. However, these techniques are not as robust as most of the digital data hiding techniques that we review in this paper. Furthermore, they cannot embed as much data in a host signal as the digital approaches.

Digital data embedding has many applications. Foremost is passive and active copyright protection. Many of the inherent advantages of digital signals increase problems associated with copyright enforcement. For this reason, creators and distributors of digital data are hesitant to provide access to their intellectual property. Digital watermarking has been proposed as a means to identify the owner or distributor of digital data.

Data embedding also provides a mechanism for embedding important control, descriptive, or reference information in a given signal. This information can be used for tracking the use of a particular clip, e.g., for pay-per-use applications, including billing for commercials and video and audio broadcast, as well as Internet electronic commerce of digital media. It can be used to track audio or visual object creation, manipulation, and modification history within a given signal without the overhead associated with creating

he knows that the host signal contains data and is familiar with the exact algorithm for embedding the data. Note that in some applications, e.g., covert communications, the data may also be encrypted prior to insertion in a host signal.

F. Copyright Protection and Ownership Deadlock

Data-embedding algorithms may be used to establish ownership and distribution of data. In fact, this is the application of data embedding or watermarking that has received most attention in the literature. Unfortunately, most current watermarking schemes are unable to resolve rightful ownership of digital data when multiple ownership claims are made, i.e., when a deadlock problem arises. The inability of many data-embedding algorithms to deal with deadlock, first described by Craver *et al.* [15], is independent of how the watermark is inserted in the multimedia data or how robust it is to various types of modifications.

Today, no scheme can unambiguously determine ownership of a given multimedia signal if it does not use an original or other copy in the detection process to at least construct the watermark to be detected. A pirate can simply add his watermark to the watermarked data or counterfeit a watermark that correlates well or is detected in the contested signal. Current data-embedding schemes used as copyright-protection algorithms are unable to establish who watermarked the data first. Furthermore, none of the current data-embedding schemes has been proven to be immune to counterfeiting watermarks that will correlate well with a given signal as long as the watermark is not restricted to depend partially in a noninvertible manner on the signal.

If the detection scheme can make use of the original to construct the watermark, then it may be possible to establish unambiguous ownership of the data regardless of whether the detection scheme subtracts the original from the signal under consideration prior to watermark detection or not. Specifically, [16] derives a set of sufficient conditions that watermarks and watermarking schemes must satisfy to provide unambiguous proof of ownership. For example, one can use watermarks derived from pseudorandom sequences that depend on the signal and the author. Reference [16] establishes that this will work for *all* watermarking procedures regardless of whether they subtract the original from the signal under consideration prior to watermark detection or not. Reference [85] independently derived a similar result for a restricted class of watermarking techniques that rely on subtracting a signal derived from the original from the signal under consideration prior to watermark detection. The signal-dependent key also helps to thwart the "mix-and-match" attack described in [16].

An author can construct a watermark that depends on the signal and the author and provides unambiguous proof of ownership as follows. The author has two random keys x_1 and x_2 (i.e., seeds) from which a pseudorandom sequence y can be generated using a suitable pseudorandom sequence generator [76]. Popular generators include RSA, Rabin, Blum/Micali, and Blum/Blum/Shub [25]. With the two

proper keys, the watermark may be extracted. Without the two keys, the data hidden in the signal are statistically undetectable and impossible to recover. Note that classical maximal length pseudonoise sequences (i.e., m -sequence) generated by linear feedback shift registers are *not* used to generate a watermark. Sequences generated by shift registers are cryptographically insecure: one can solve for the feedback pattern (i.e., the keys) given a small number of output bits y .

The noise-like sequence y may be used to derive the actual watermark hidden into the signal or to control the operation of the watermarking algorithm, e.g., to determine the location of pixels that may be modified. The key x_1 is *author* dependent. The key x_2 is *signal* dependent. The key x_1 is the secret key assigned to (or chosen by) the author. The key x_2 is *computed from the signal* that the author wishes to watermark. It is computed from the signal using a one-way hash function. For example, the tolerable error levels supplied by masking models (see Section IV) are hashed in [85] to a key x_2 . Any one of a number of well-known secure one-way hash functions may be used to compute x_2 , including RSA, MD4 [77], and SHA [60]. For example, the Blum/Blum/Shub pseudorandom generator uses the one-way function $y = g_n(x) = x^2 \bmod n$, where $n = pq$ for primes p and q so that $p = q = 3 \bmod 4$. It can be shown that generating x or y from partial knowledge of y is *computationally infeasible* for the Blum/Blum/Shub generator.

The signal-dependent key x_2 makes counterfeiting very difficult. The pirate can only provide key x_1 to the arbitrator. Key x_2 is automatically computed by the watermarking algorithm from the original signal. As it is computationally infeasible to invert the one-way hash function, the pirate is unable to fabricate a counterfeit original that generates a desired or predetermined watermark.

Deadlock may also be resolved using the dual watermarking scheme of [85]. That scheme employs a *pair* of watermarks. One watermarking procedure requires the original data set for watermark detection. The second watermarking procedure does *not* require the original data set. A data-embedding technique that satisfies the restrictions outlined in [16] can be used to insert the second watermark.

The above discussion clearly highlights the limitation of watermarking as an unambiguous mean of establishing ownership. Future clever attacks may show that the schemes described in [16] or [85] are still vulnerable to deadlock. Furthermore, all parties would need to use watermarking techniques that have been proven or certified to be immune to deadlock to establish ownership of media. Note also that contentions of ownership can occur in too many different forms. Copyright protection will probably not be resolved exclusively by one group or even the entire technical community since it involves too many legal issues, including the very definition of similarity and derived works. Many multidisciplinary efforts are currently investigating standards and rules for national and international copyright protection and enforcement in the digital age.

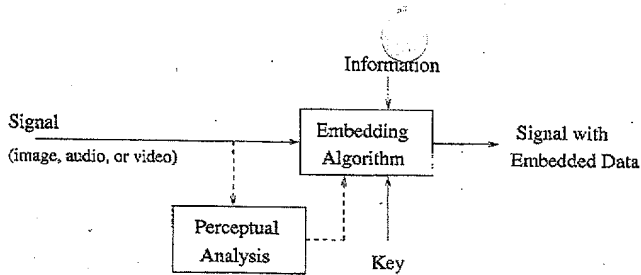


Fig. 1. Diagram of a data-embedding algorithm. The information is embedded into the signal using the embedding algorithm and a key. The dashed lines indicate that the algorithm may directly exploit perceptual analysis to embed information.

IV. SIGNAL INSERTION: THE ROLE OF MASKING

The first problem that all data-embedding and watermarking schemes need to address is that of inserting data in the digital signal without deteriorating its perceptual quality. Of course, we must be able to retrieve the data from the edited host signal, i.e., the insertion method must also be invertible. Since the data-insertion and data-recovery procedures are intimately related, the insertion scheme must take into account the requirement of the data-embedding application. In many applications, we will need to be able to retrieve the data even when the host signal has undergone modifications, such as compression, editing, or translation between formats, including A/D and D/A conversions.

Data insertion is possible because the digital medium is ultimately consumed by a human. The human hearing and visual systems are imperfect detectors. Audio and visual signals must have a minimum intensity or contrast level before they can be detected by a human. These minimum levels depend on the spatial, temporal, and frequency characteristics of the human auditory and visual systems. Further, the human hearing and visual systems are characterized by an important phenomenon called masking. Masking refers to the fact that a component in a given audio or visual signal may become imperceptible in the presence of another signal called the masker. Most signal-coding techniques (e.g., [41]) exploit the characteristics of the human auditory and visual systems directly or indirectly. Likewise, all data-embedding techniques exploit the characteristics of the human auditory and visual systems implicitly or explicitly (see Fig. 1). In fact, embedding data would not be possible without the limitations of the human visual and auditory systems. For example, it is not possible to modify a binary stream that represents programs or numbers that will be interpreted by a computer. The modification would directly and adversely affect the output of the computer.

A. The Human Auditory System (HAS)

Audio masking is the effect by which a faint but audible sound becomes inaudible in the presence of another louder audible sound, i.e., the masker [42]. The masking effect depends on the spectral and temporal characteristics of both the masked signal and the masker.

Frequency masking refers to masking between frequency components in the audio signal. If two signals that occur simultaneously are close together in frequency, the stronger masking signal may make the weaker signal inaudible. The

masking threshold of a masker depends on the frequency, sound pressure level, and tone-like or noise-like characteristics of both the masker and the masked signal [61]. It is easier for a broad-band noise to mask a tonal signal than for a tonal signal to mask out a broad-band noise. Moreover, higher frequency signals are more easily masked.

The human ear acts as a frequency analyzer and can detect sounds with frequencies that vary from 10 to 20 000 Hz. The HAS can be modeled by a set of bandpass filters with bandwidths that increase with increasing frequency. The bands are known as the critical bands. The critical bands are defined around a center frequency in which the noise bandwidth is increased until there is a just noticeable difference in the tone at the center frequency. Thus, if a faint tone lies in the critical band of a louder tone, the faint tone will not be perceptible.

Frequency-masking models are readily obtained from the current generation of high-quality audio codecs, e.g., the masking model defined in the International Standards Organization (ISO)-MPEG Audio Psychoacoustic Model 1 for Layer I [40]. The Layer I masking method is summarized as follows for a 32-kHz sampling rate. The MPEG model also supports sampling rates of 44.1 and 48 kHz.

The frequency mask is computed on localized segments (or windows) of the audio signal. The first step consists of computing the power spectrum of a short window (512 or 1024 samples) of the audio signal. Tonal (sinusoidal) and nontonal (noisy) components in the spectrum are identified because their masking models are different. A tonal component is a local maximum of the spectrum. The auditory system behaves as a bank of bandpass filters, with continuously overlapping center frequencies. These "auditory filters" can be approximated by rectangular filters with critical bandwidth increasing with frequency. In this model, the audible band is therefore divided into 24 nonregular critical bands.

Next, components below the absolute hearing threshold and tonal components separated by less than 0.5 Barks are removed. The final step consists of computing individual and global masking thresholds. The frequency axis is discretized according to hearing sensitivity and express frequencies in Barks. Note that hearing sensitivity is higher at low frequencies. The resulting masking curves are almost linear and depend on a masking index different for tonal and nontonal components. They are characterized by different lower and upper slopes depending on the distance between the masked and the masking component. We use f_1 to denote the set of frequencies present in the test signal. The global masking threshold for each frequency f_2 takes into account the absolute hearing threshold S_a and the masking curves P_2 of the N_t tonal components and N_n nontonal components

$$S_m(f_2) = 10 * \log_{10} \left[10^{S_a(f_2)/10} + \sum_{j=1}^{N_t} 10^{P_2(f_2, f_1, P_1)/10} + \sum_{j=1}^{N_n} 10^{P_2(f_2, f_1, P_1)/10} \right] \quad (1)$$

AFFIDAVIT OF JOHN Q. PUBLIC

John Q. Public declares as follows:

1. I have been presented with a ZIP file entitled FILES.ZIP containing, among other files, an executable file entitled HASH.EXE and a document in Adobe Acrobat format entitled DOCUMENT.PDF. After extracting HASH.EXE from FILES.ZIP and launching it, I opened FILES.ZIP in the program window that was created upon launch. I then saw a display appear in the program window with the legend "160-bit Hash Value (SHA1)" and a string of alphanumeric characters. The characters were identical to those printed below and repeated throughout the background of this entire paper as vertically oriented digits in various outline fonts.

BD7D F2FD EC1C DF14 4811 574F F7CE 7D1E 6EB6 F7E9

2. I extracted DOCUMENT.PDF from FILES.ZIP and printed out the document reproduced therein. I took the printed copy into the file inspection room of the U.S. Patent Office and compared each page printed from DOCUMENT.PDF with the papers physically mounted in the official file of U.S. Patent 6,052,748.

3. I personally confirmed that each page of DOCUMENT.PDF is a true and correct copy of a page in the file of U.S. Patent 6,052,748. I further confirmed that all pages of the file have been reproduced in DOCUMENT.PDF.

I declare under penalty of perjury that the foregoing statements are true and correct.

Signed at _____, This _____ day of _____,
City State Date

By: **VOID**
John Q. Public

This _____ day of _____, _____, the above-named person personally came before me and executed the foregoing Affidavit in my presence.

Notary Public: **VOID**

Notary: Outline digits are in background of entire document. Please stamp over digits in space above.

AFFIDAVIT OF JAMES T. GRIFFIN

James T. _____ declares as follows:

1. I have been presented with a ZIP file entitled 1033.ZIP containing, among other files, another ZIP file entitled 1033-SRC.ZIP, which contains an executable program file entitled GPG.EXE and a document in Adobe Acrobat format entitled 1033-ART.PDF. After extracting GPG.EXE from 1033-SRC.ZIP, I executed it from the Windows 98/NT/2000 command line with the following text:

gpg --print-md sha1 1033.zip

I then saw a display appear in the program window with a string of alphanumeric characters. The characters were identical to those printed below this paragraph and repeated throughout the background of this entire paper as vertically oriented digits in various outline fonts.

8D06 7D30 7CDD 969E 3C3C C57D E9E3 467D 2E54 6AC3

2. I opened 1033-ART.PDF, which was also extracted from 1033.ZIP, and viewed it on my computer screen. I carefully compared the displayed document with an original artwork having images of flowers formed from torn-out pieces of paper, fixed to a piece of construction paper beneath a transparent laminating surface.

3. I personally confirmed that 1033-ART.PDF is a true and correct reproduction of the original artwork, to the degree of accuracy permitted by the displayed reproduction. The only visible discrepancies between the reproduction and the original artwork are (1) the coloration of the background, which appears as a washed-out green in the original artwork and has more of a purplish hue in the reproduction, and (2) the lower 1 inch or so of the original artwork is not shown in the reproduction.

I declare under penalty of perjury that the foregoing statements are true and correct.

Signed at _____ This _____ day of _____
City State Date

By: VOID
James T. _____

This _____ day of _____, the above-named person personally came before me and executed the foregoing Affidavit in my presence.

Notary Public: VOID

Notary: Outline digits are in background of entire document. Please stamp over digits in space above.

APPENDIX H

FILING CONTENTS AUTHENTICATION

Joe Agent creates a document when filing a patent application for a client. He wants an additional piece of evidence that the patent application was mailed and that what was mailed was the patent application. So, he creates a PDF file of the application's text and drawings and computes the SHA1 hash of the PDF file using some the free GPG software ("GNU Privacy

Guard"), which he trusts. He makes a copy of the Express Mail label under which the application is to be mailed BEFORE having the label initialed and dated by the postal worker.

He then makes a Word 97 document that includes a brief block of text for a witness to sign and date. The block of text includes the Express Mail label and the hash. The document displays the digits of the hash as vertically oriented digits in various outline fonts, as

with the ACI (See, e.g., Appendix A.) He then puts the photocopy of the non-initialed express

mail label in his printer's paper supply, and prints the Word 97 (RTM) document.

The resulting document bears the image of the non-initialed express mail label with the hash of the patent application PDF file throughout its background and in a block of text of the top. Joe presents the document to Jane Attorney, an attorney in the office next door, for her dated signature. Now he can present the PDF file and the signed and dated paper document as evidence that the patent application reproduced in the PDF file existed before the express mail label was used to mail a package. This provides evidence that the patent application, reproduced in the PDF file, was actually what was mailed on the date of the express mail label. The proof can be made a bit

stronger by including the express mail label number in a footer on the first page of the patent application, so that the PDF file contains the express mail label number shown in the document. But that's going way beyond the level of evidence currently needed to reconstruct a file lost by the PTO.

This particular embodiment of the inventions can be employed any situation where proof of mailing of a particular document is desired, without the need for digital signatures and the associated hassles of public key authentication. The witness to this document doesn't even

need to know what digital signatures or hash codes are. He or she is simply testifying as to the existence of the document with that particular code.

2050ED-E462600T

$$f(x, y, N) = \left(\frac{x - y}{x} \right)^N$$

$$f(x, y, N) = \exp \left(N \cdot \ln \left(\left| 1 - \frac{1}{x} \cdot y \right| \right) \right) \cdot \cos \left[N \cdot \left(\frac{1}{2} - \frac{1}{2} \cdot 1 \right) \cdot \pi \right] \dots$$

$$+ i \cdot \exp \left(N \cdot \ln \left(\left| 1 - \frac{1}{x} \cdot y \right| \right) \right) \cdot \sin \left[N \cdot \left(\frac{1}{2} - \frac{1}{2} \cdot 1 \right) \cdot \pi \right]$$

$$f(x, y, N) = \exp \left(N \cdot \ln \left(\left| 1 - \frac{1}{x} \cdot y \right| \right) \right) \cdot 1 \dots$$

$$+ i \cdot \exp \left(N \cdot \ln \left(\left| 1 - \frac{1}{x} \cdot y \right| \right) \right) \cdot \sin \left[N \cdot \left(\frac{1}{2} - \frac{1}{2} \cdot 1 \right) \cdot \pi \right]$$

$$f(x, y, N) := \exp \left(N \cdot \ln \left(\left| 1 - \frac{1}{x} \cdot y \right| \right) \right)$$

M

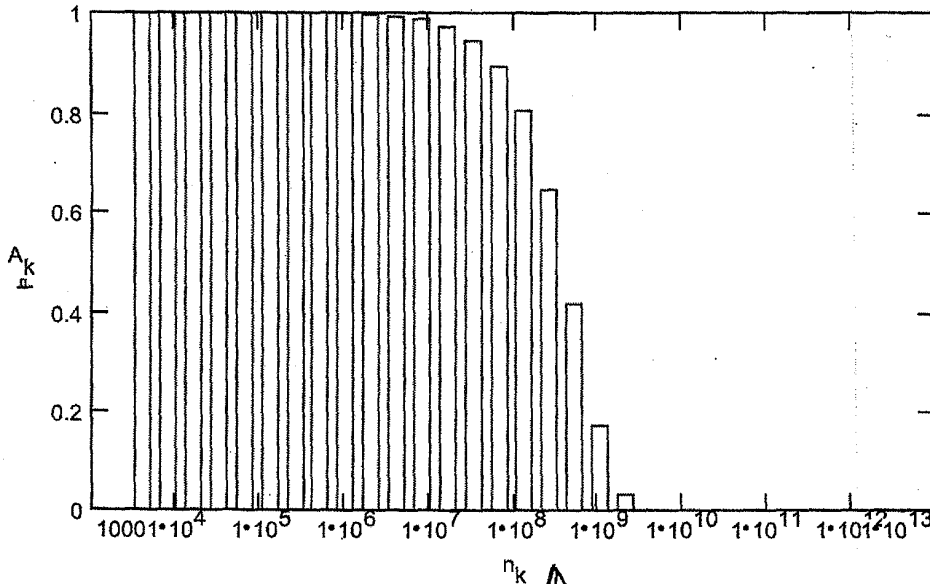
$$x := 2^{32} \quad y := 7 \quad \leftarrow \rho = 2^{32} - 7$$

$$k := 12..40 \quad n_k := 2^k$$

$$A_k := f(x, y, n_k)$$

output
changed, i.e.,
not a

Graph: Product of probabilities of obtaining pass-through value, for each iteration, given n iterations.



\$n_k\$ ↑
50% chance of
getting one pass-through
value w/ random inputs
after \$10^8 - 10^9\$ iterations.

$$x := 2^{50}$$

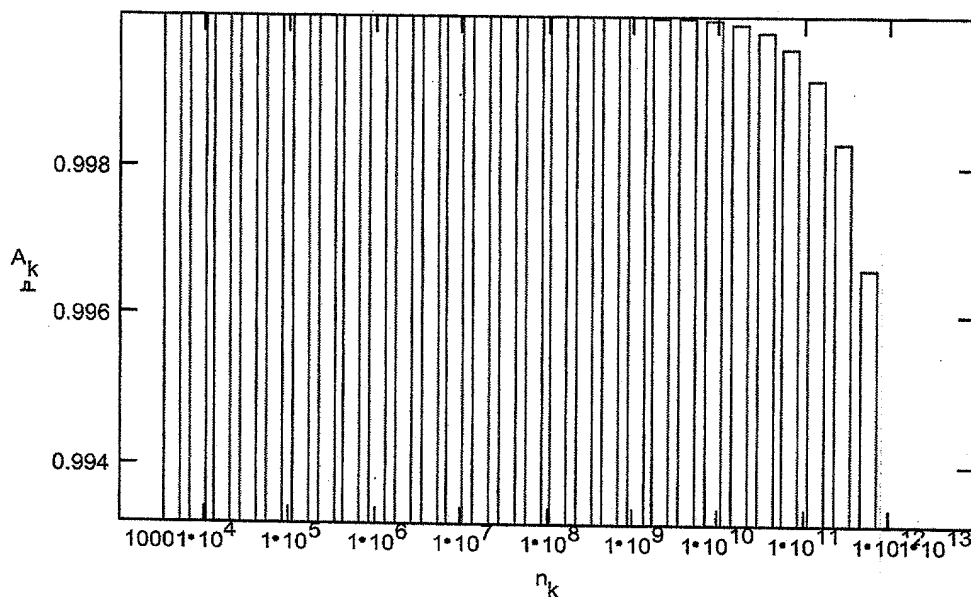
$$A_k := f(x, y, n_k)$$

Larger M
Same k

changed output, i.e.,
not q

Graph: Product of probabilities of obtaining pass-through value, for each iteration, given n iterations.

$$p = M - k$$



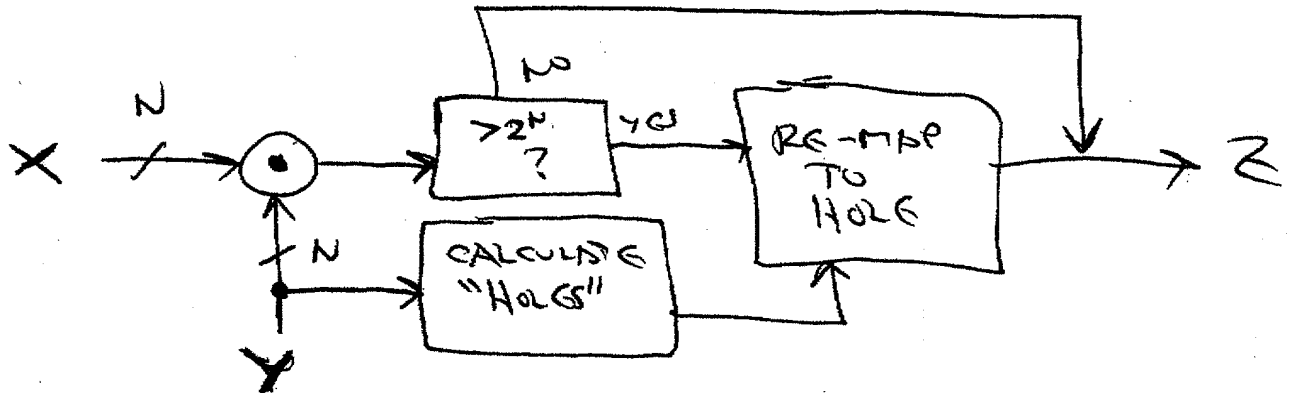
Need for exception
handling can be made
vanishingly unlikely.

With $p > 2^N$:

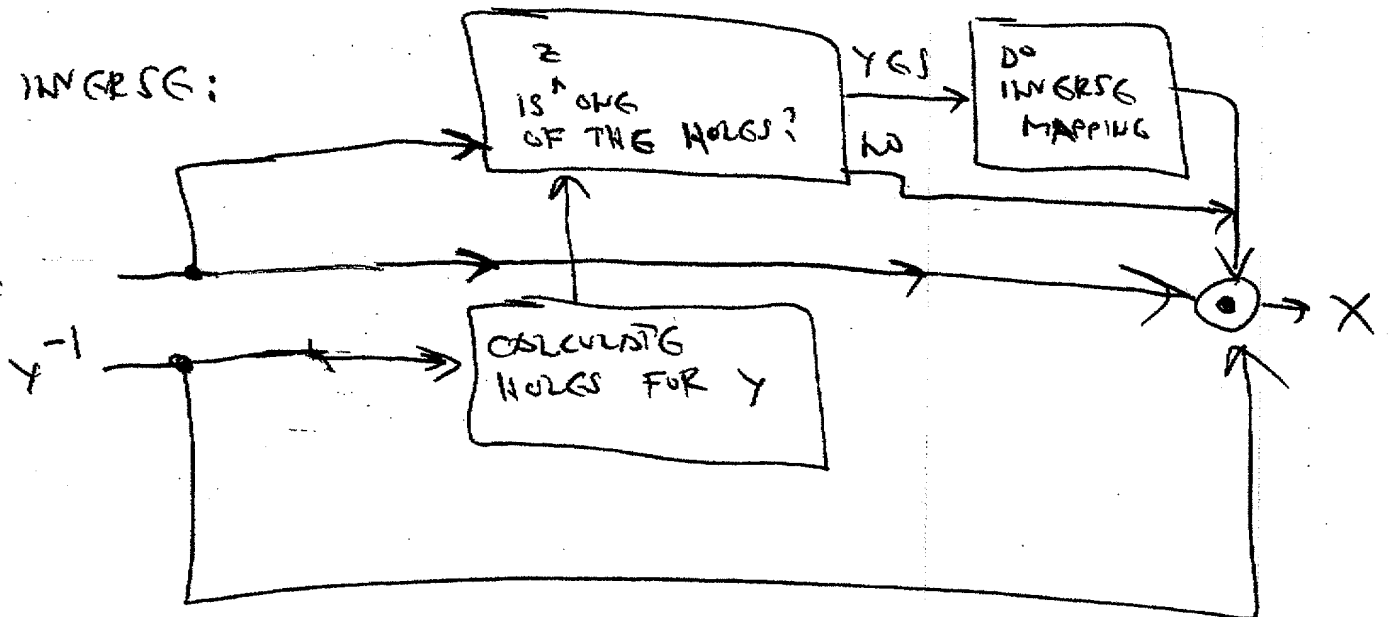
MODIFIED "⊙"

To work OUTSIDE

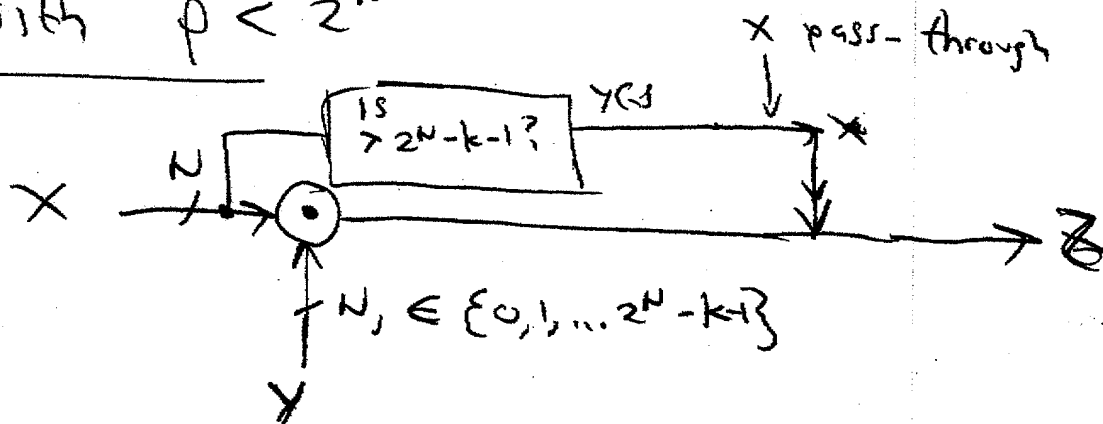
$$N = \{4, 8, 16\}, p = 2^N + 1$$



INVERSE:



With $p < 2^N$



Interesting Primes (closest to 2^N)

$$2^{20} + 7$$

$$2^{20} - 3$$

$$2^{32} + 15$$

$$2^{32} - 5$$

$2^N + 1$ not prime for $16 < N \leq 32$

$$2^{16} + 3, -15$$

$$2^{18} + 3, -5$$

$$2^{19} - 1$$

$$2^{28} + 3$$

Bit

Length

Test Number (Y=prime, N=not prime, blank=not tested)

	2^N-1	2^N-3	2^N-5	2^N-7	2^N-9	2^N-11	2^N-13	2^N-15	2^N-17
16	N	N	N	N	N	N	N	N	
17	Y	N							
18	N	N							
19	Y								
20	N	Y							
21	N	N							
22	N	Y							
23	N	N							
24	N	Y							
25	N	N							
26	N	N							
27	N	N							
28	N	N							
29	N	Y							
30	N	N							
31	Y								
32	N	N	Y						
33	N	N	N	N	N				
34	N	N	N	N	N				
35	N	N	N	N	N				
36	N	N	Y						
37	N	N	N	N	N				
38	N	N	N	N	N				
39	N	N	N	Y					
40	N	N	N	N	N	N	N	N	N
41	N	N	N	N	N	N			
42	N	N	N	N	N	N			
43	N	N	N	N	N	N			
44	N	N	N	N	N	N			
45	N	N	N	N	N	N			
46	N	N	N	N	N	N			
47	N	N	N	N	N	N			
48	N	N	N	N	N	N	N	N	N

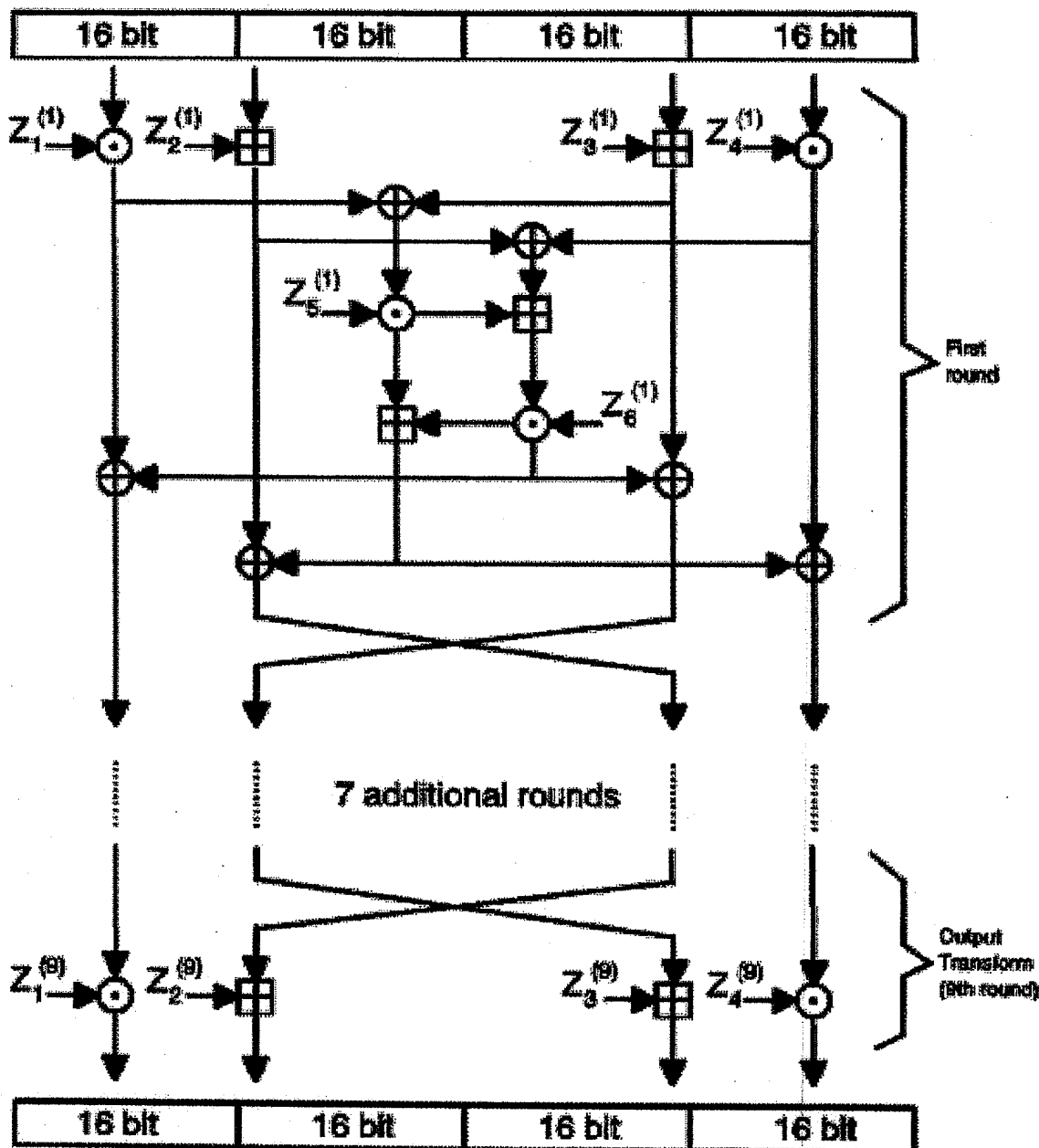
Likelihood
of pass-through
is $6/2^{32}$
 $= 1.4 \times 10^{-9}$
(very infrequent)

Effective bit length for one "leaked" bit, per offset below 2^N

2^N	1	3	5	7	9	11	13	15	17
65536									
131072	16.0							12.0	
262144									
524288	18.0								
1048576		18.0							
2097152			20.0						
4194304				22.0					
8388608									
16777216									
33554432									
67108864									
134217728									
268435456									
536870912		27.0							
1073741824									
2147483648	30.0								
4294967296		29.4							
8589934592									
17179869184									
34359738368									
68719476736									
137439E+11									
274878E+11									
549756E+11									
109951E+12									
219902E+12									
439805E+12									
879609E+12									
175922E+13									
351844E+13									
703687E+13									
140737E+14									
281475E+14									

Six
five members of $\{0,1\}^{32}$
(4,294,967,290; 4...291; 4...292;
4...293; 4...294; 4...295)
are passed through function w/o
modification.

Plaintext (64 bit)



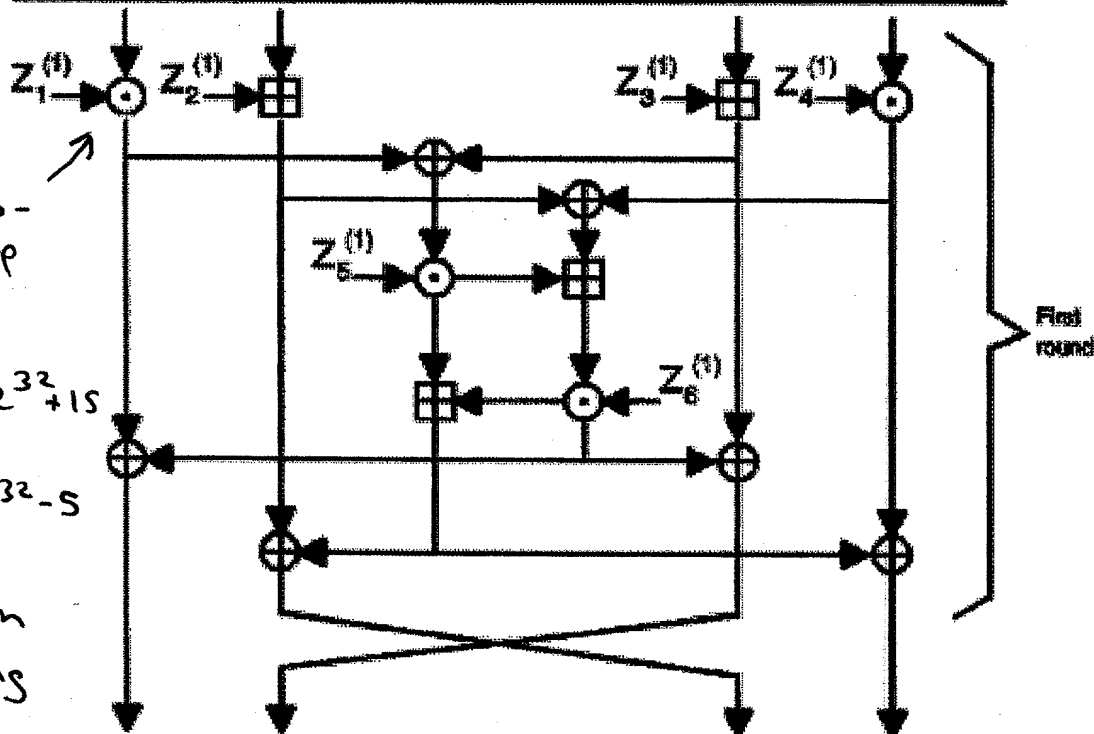
Ciphertext (64 bit)

- \oplus Bit-by-bit exclusive OR of two 16-bit subblocks
- \boxplus Addition modulo 2^{16} of two 16-bit integers
- \odot Multiplication modulo $2^{16} + 1$ of two 16-bit integers (subblock of all zeroes corresponds to 2^{16})

IDEA CIPHER

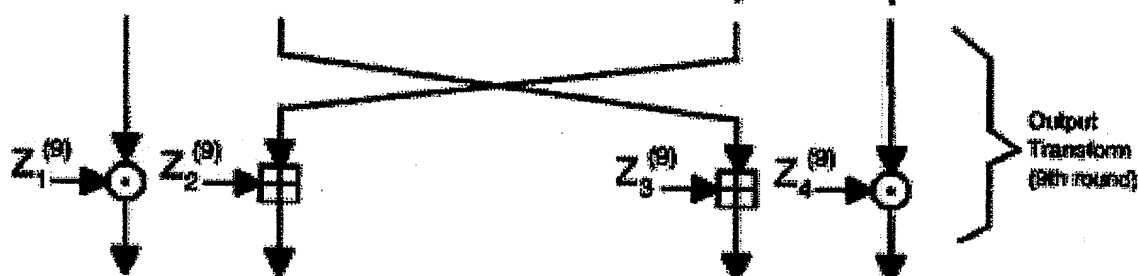
From H.A.C. (Handbook of Applied Cryptography)

Plaintext (128 bit)



First round

7 additional rounds



Output Transform (8th round)



Ciphertext (128 bit)

- \oplus Bit-by-bit exclusive OR of two 32-bit subblocks
- \boxplus Addition modulo 2^{32} of two 32-bit integers
- \odot Multiplication modulo $2^{32}-5$ of two 32-bit integers

Adapted From H.A.C.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2	4	6	8	10	12	1	3	5	7	9	11	13	14	15	16
3	3	6	9	12	2	5	8	11	1	4	7	10	13	14	15	16
4	4	8	12	3	7	11	2	6	10	1	5	9	13	14	15	16
5	5	10	2	7	12	4	9	1	6	11	3	8	13	14	15	16
6	6	12	5	11	4	10	3	9	2	8	1	7	13	14	15	16
7	7	1	8	2	9	3	10	4	11	5	12	6	13	14	15	16
8	8	3	11	6	1	9	4	12	7	2	10	5	13	14	15	16
9	9	5	1	10	6	2	11	7	3	12	8	4	13	14	15	16
10	10	7	4	1	11	8	5	2	12	9	6	3	13	14	15	16
11	11	9	7	5	3	1	12	10	8	6	4	2	13	14	15	16
12	12	11	10	9	8	7	6	5	4	3	2	1	13	14	15	16
13	ILLEGAL KEY VALUES															
14																
15																
16																

$$N = 4, \phi = 2^N - 3$$

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	4	6	1	3	5	7	8
3	3	6	2	5	1	4	7	8
4	4	1	5	2	6	3	7	8
5	5	3	1	6	4	2	7	8
6	6	5	4	3	2	1	7	8
7	ILLEGAL KEY VALUES							
8								

$$N = 3, \phi = 2^N - 1$$

1	2	3	...	20,537	20,538	20,539	...	16,777,211	16,777,212	16,777,213	16,777,214	16,777,215	16,777,216
1	2	3	...	20,537	20,538	20,539	...	16,777,211	16,777,212	16,777,213	16,777,214	16,777,215	16,777,216
2	4	6	...	41,074	41,076	41,078	...	16,777,209	16,777,211	16,777,213	16,777,214	16,777,215	16,777,216
3	6	9	...	61,611	61,614	61,617	...	16,777,207	16,777,210	16,777,213	16,777,214	16,777,215	16,777,216
...
20,537	41,074	61,611	...	2,338,044	2,358,581	2,378,118	...	16,736,139	16,756,676	16,777,213	16,777,214	16,777,215	16,777,216
20,538	41,076	61,614	...	2,358,581	2,378,119	2,398,657	...	16,736,137	16,756,675	16,777,213	16,777,214	16,777,215	16,777,216
20,539	41,078	61,617	...	2,378,118	2,398,657	2,420,196	...	16,736,135	16,756,674	16,777,213	16,777,214	16,777,215	16,777,216
...
16,777,211	16,777,209	16,777,207	...	16,736,139	16,736,137	16,736,135	...	4	2	16,777,213	16,777,214	16,777,215	16,777,216
16,777,212	16,777,211	16,777,210	...	16,756,676	16,756,675	16,756,674	...	2	1	16,777,213	16,777,214	16,777,215	16,777,216
16,777,213													
16,777,214													
16,777,215													
16,777,216													

ILLEGAL KEY VALUES

$$N = 24, \varphi = 2^N - 3$$

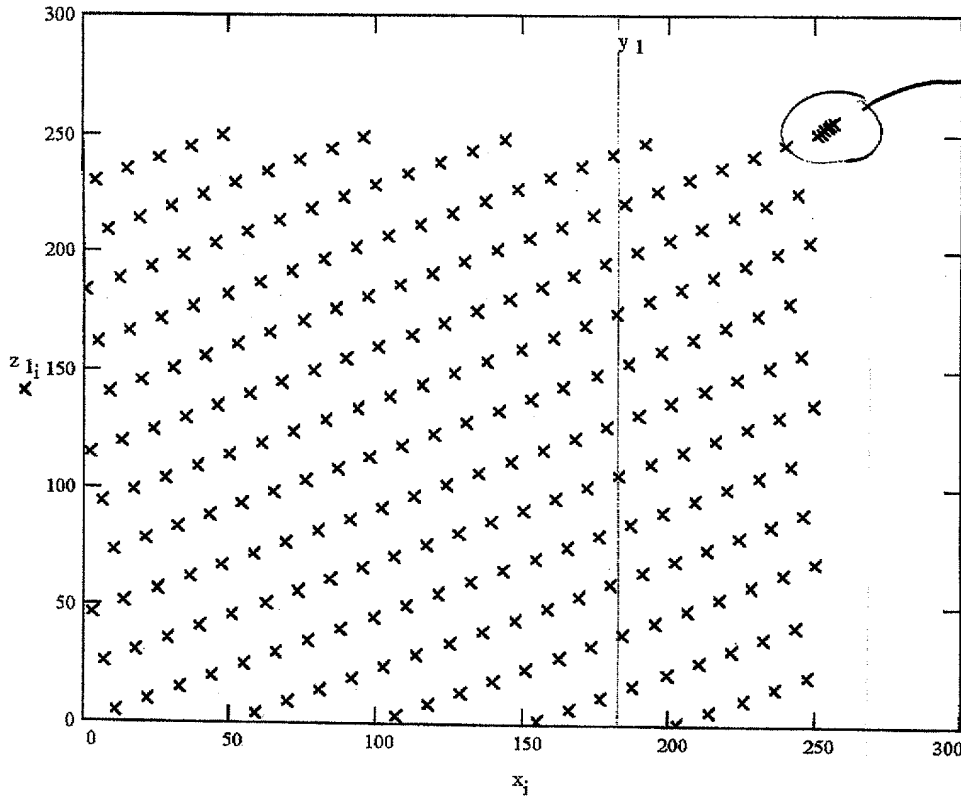
Multiplicative Group Operation (pseudogroup)

$$f(x,y) = xy \bmod 2^N - k, \text{ where } x, y \in \mathbb{Z}_{2^N - k};$$

$$= x \text{ where } x < 2^N - k. (y \text{ always } < 2^N - k.)$$

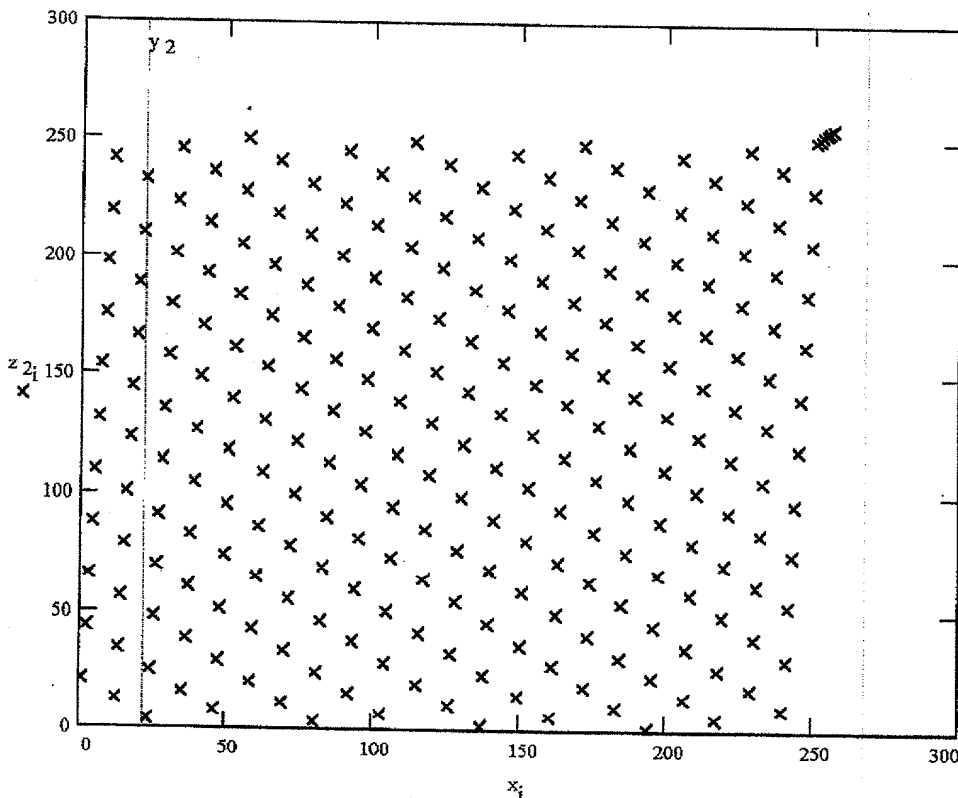
N = 8 bits $2^N = 256$ k = 5 p = 251 prime modulus

Pseudogroup operation with key y1: $y_1 = 183$ $z_{1,i} := f(x_i, y_1)$



Linear
exception
region
per EBS
'invention'
where
 $x > p-1$

Pseudogroup operation with key y2: $y_2 = 22$ $z_{2,i} := f(x_i, y_2)$



Multiplicative Group Operation (Pseudogroup)

$f(x,y) = xy \bmod 2^N - k$, where $x, y < 2^N - k$;
 $= x$ where $x < 2^N - k$. (y always $< 2^N - k$.)

$N = 8$ bits

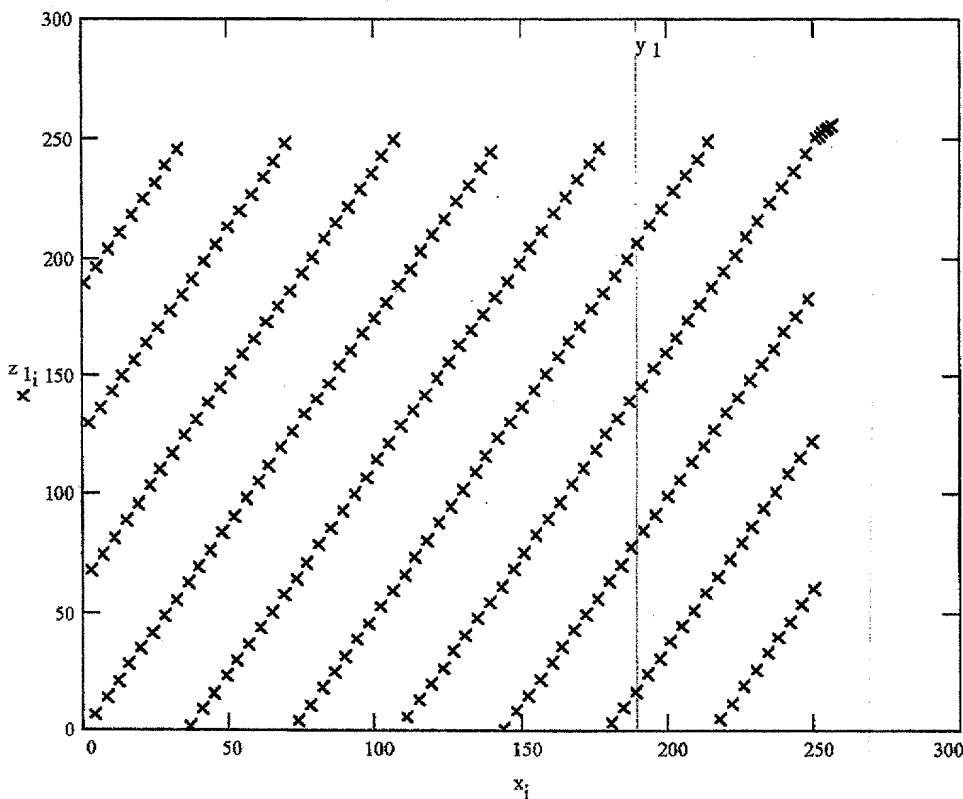
$2^N = 256$

$k = 5$

$p = 251$

prime modulus

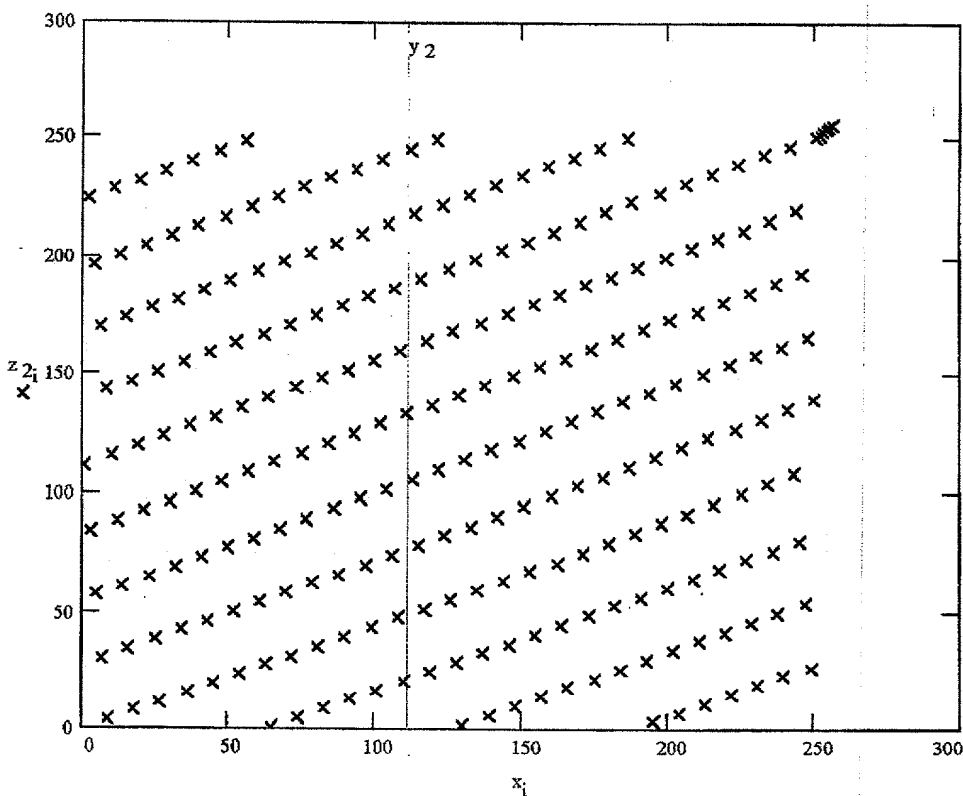
Pseudogroup operation with key y_1 : $y_1 = 190$ $z_{1,i} := f(x_i, y_1)$



Pseudogroup operation with key y_2 :

$y_2 = 112$

$z_{2,i} := f(x_i, y_2)$



Multiplicative Group Operation (pseudogroup)

$$f(x,y) = xy \bmod 2^N - k, \text{ where } x, y \in \mathbb{Z}_{2^N - k};$$

$$= x \text{ where } x < 2^N - k. \text{ (y always } < 2^N - k.)$$

$N = 8$ bits

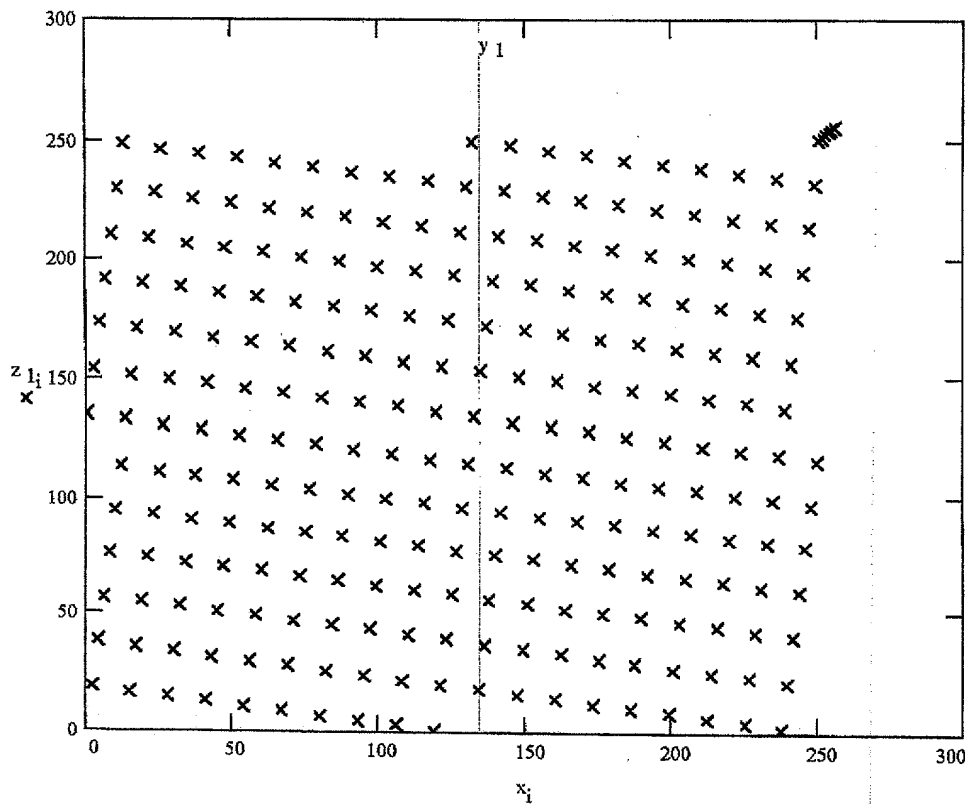
$$2^N = 256$$

$k = 5$

$p = 251$

prime modulus

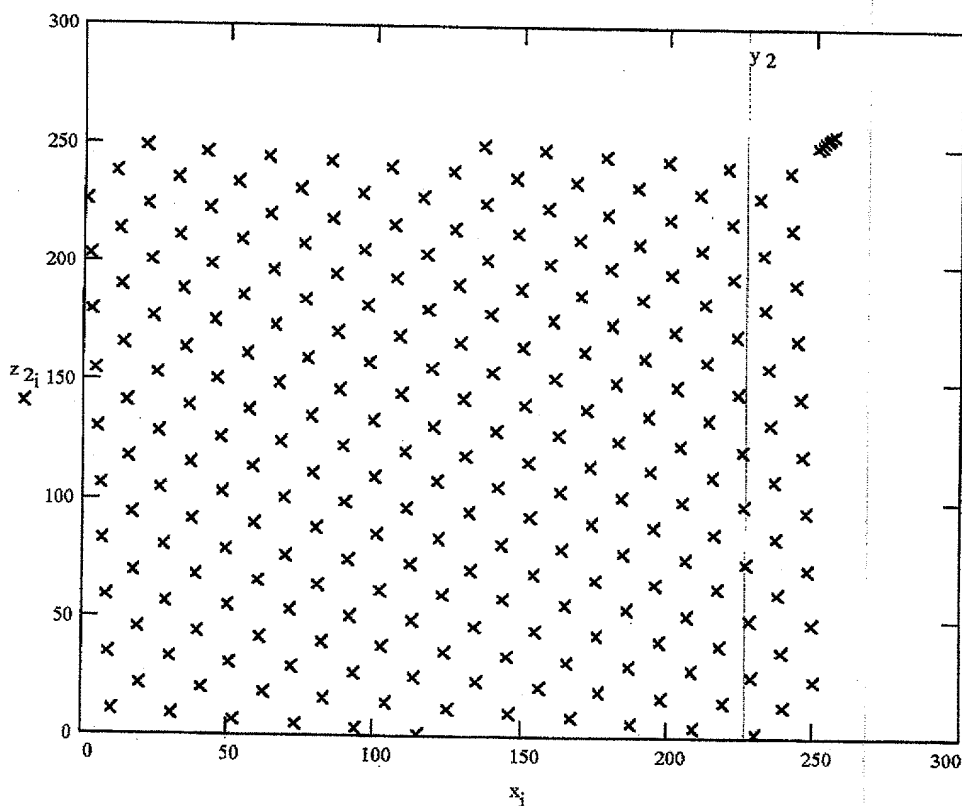
Pseudogroup operation with key y_1 : $y_1 = 135$ $z_{1,i} := f(x_i, y_1)$



Pseudogroup operation with key y_2 :

$$y_2 = 227$$

$$z_{2,i} := f(x_i, y_2)$$



Multiplicative Group Operation (pseudonym)
 $f(x,y) = xy \bmod 2^N - k$, where $x,y \in \mathbb{N} - k$;
 $= x$ where $x < 2^N - k$. (y always $< 2^N - k$.)
 $2^N = 256$
 $k = 5$
 $p = 251$
prime modulus
 $z_1 := f(x_1, y_1)$

$N=8$ bits

$$2^N = 256$$
 $k=5$


p = 251

prime modulus

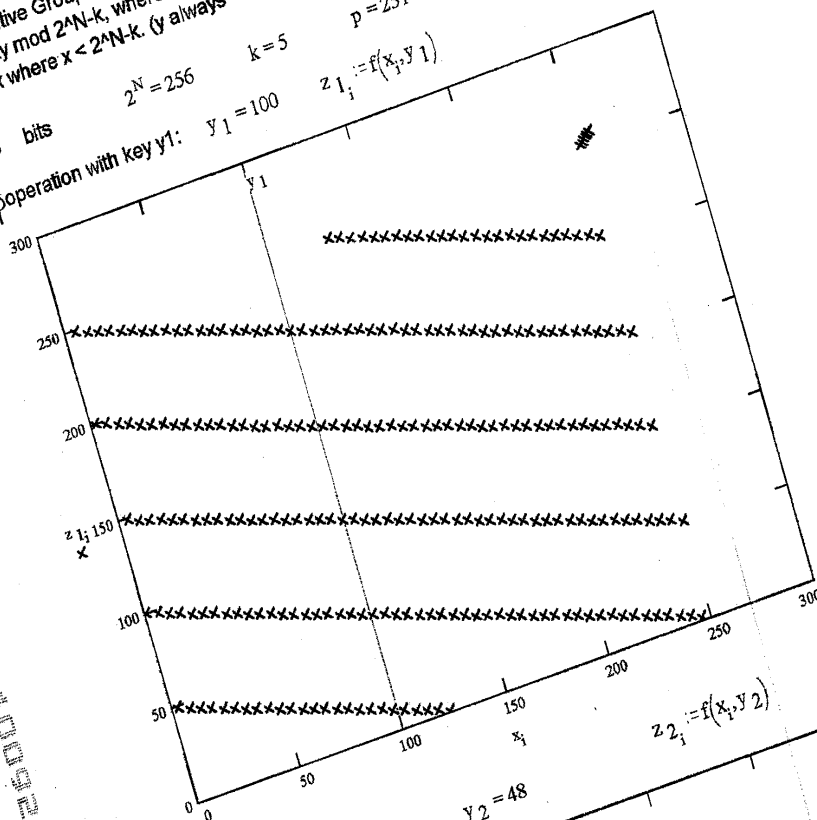
$$z_1 := f(x_1, y_1)$$

$N = 8$ bits

Pseudogroup operation with key y_1 :

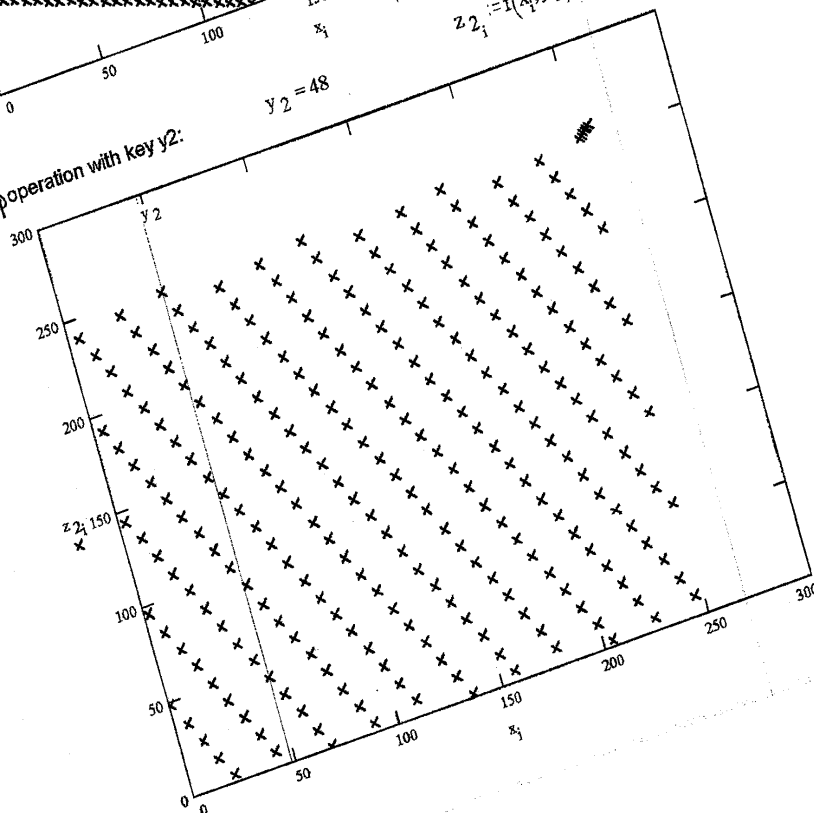


300 1



10092443 000502
PSC

pseudo group operation with key y2:



Multiplicative Group Operation (pseudogroup)

$$f(x,y) = xy \bmod 2^N - k, \text{ where } x, y < 2^N - k;$$

$$= x \text{ where } x < 2^N - k. (y \text{ always } < 2^N - k.)$$

$N = 16$ bits

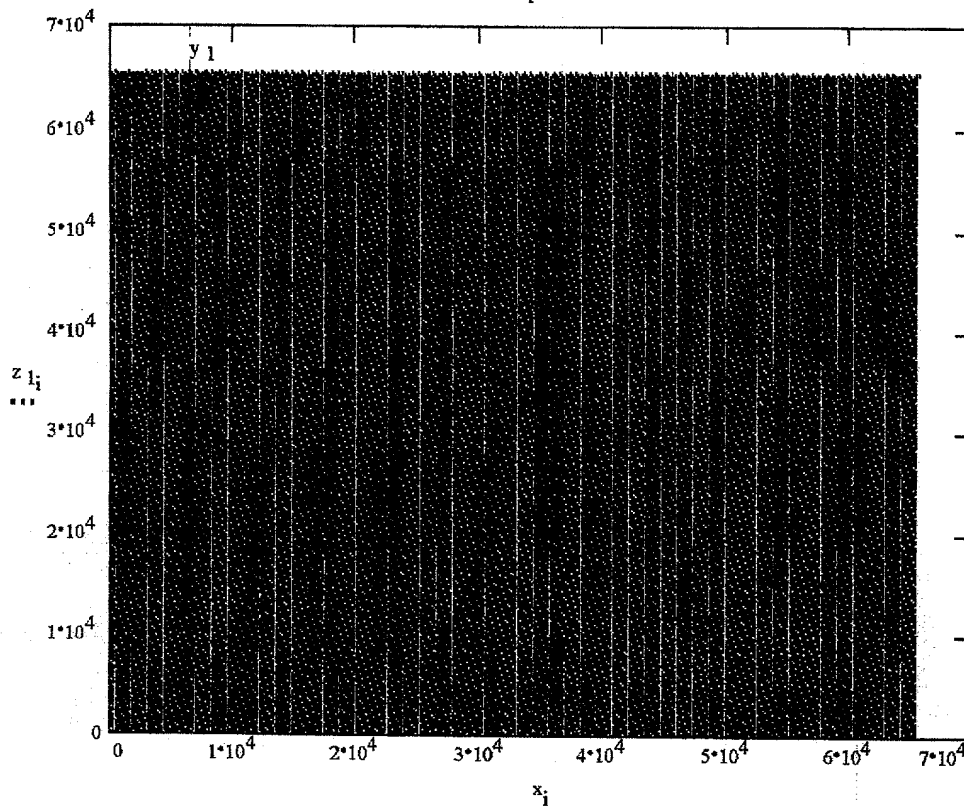
$2^N = 65536$

$k = 15$

$p = 65521$

prime modulus

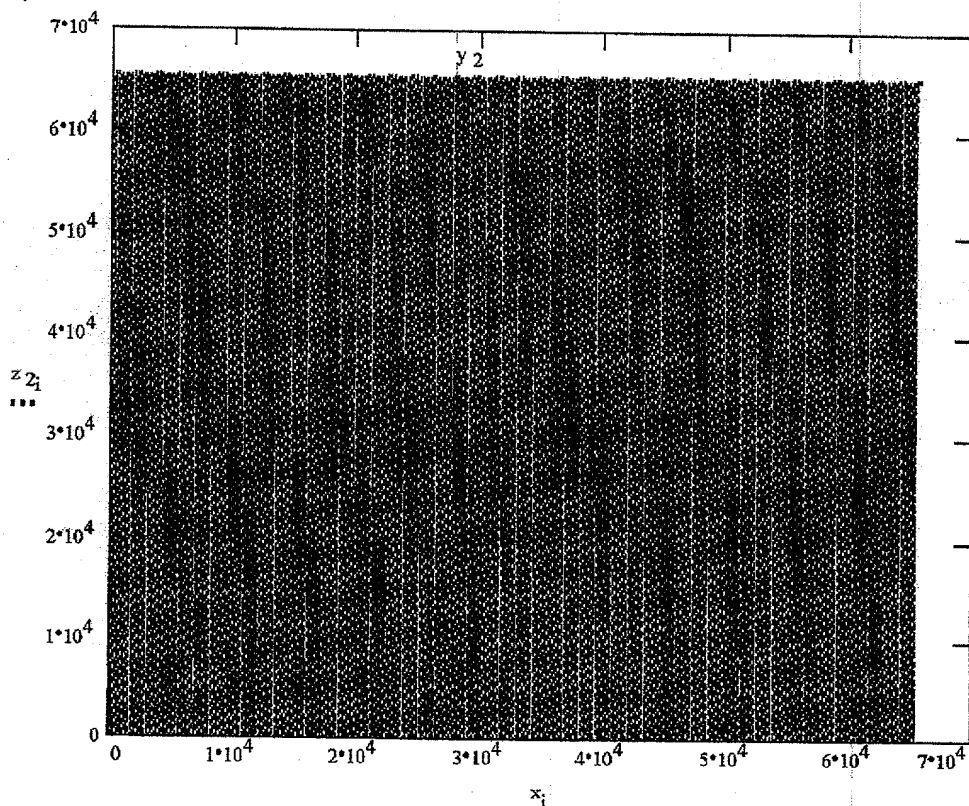
pseudogroup operation with key y_1 : $y_1 = 6595$ $z_{1_i} := f(x_i, y_1)$



pseudogroup operation with key y_2 :

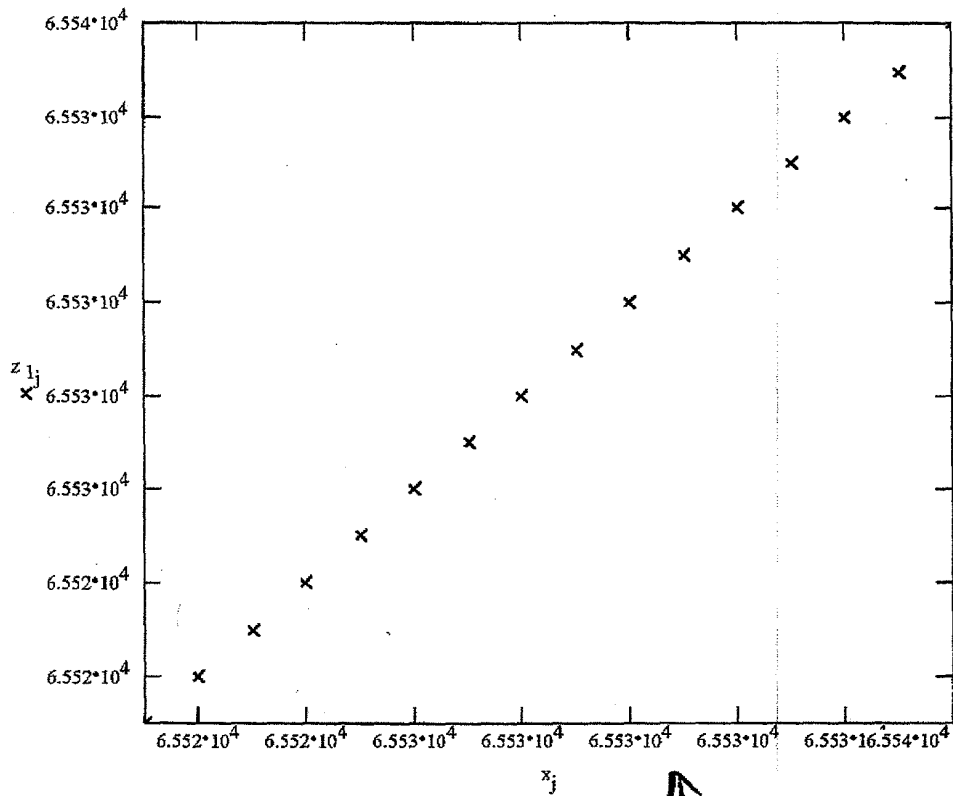
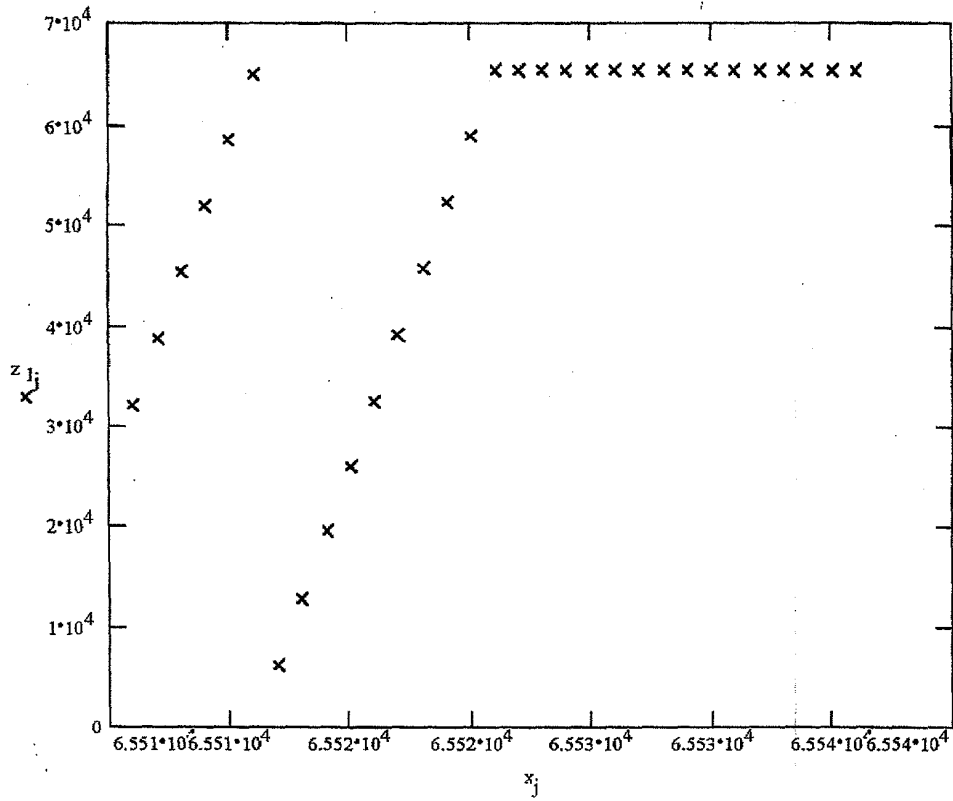
$y_2 = 27939$

$z_{2_i} := f(x_i, y_2)$



Zoom in on exception region of plot with key Y1

$$j = 2^N - 2 \cdot k \cdot 2^N$$



↑
Linear Exception
Region

205080-24626007

10092643-030503

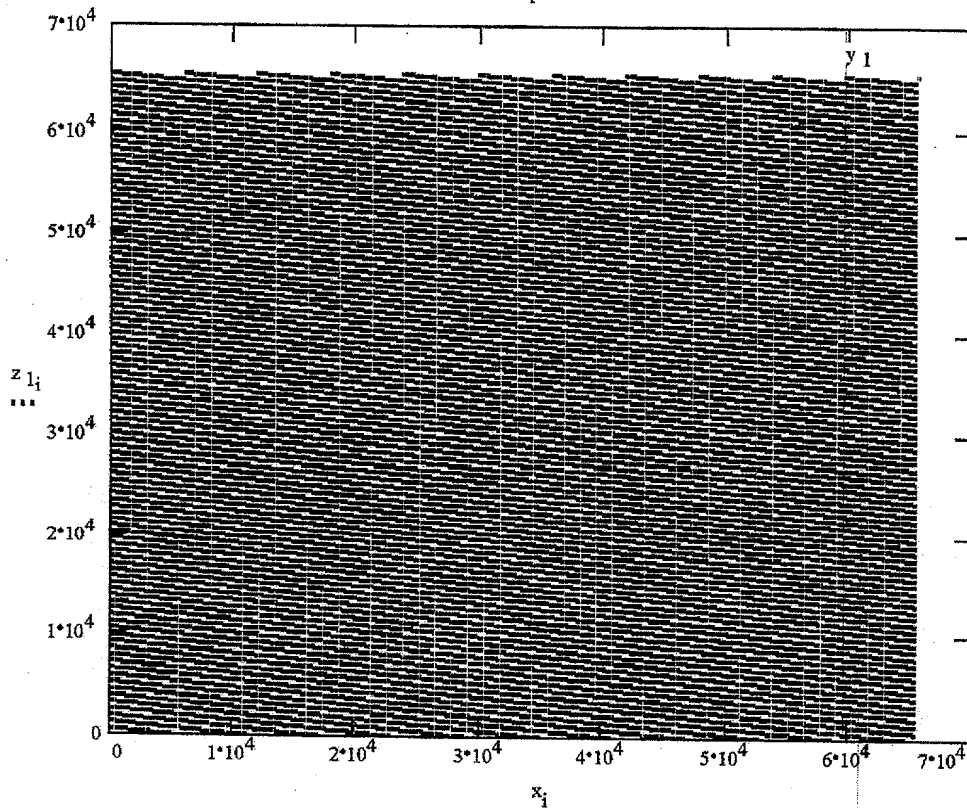
Multiplicative Group Operation (pseudogroup)

$$f(x,y) = xy \bmod 2^N - k, \text{ where } x, y < 2^N - k;$$

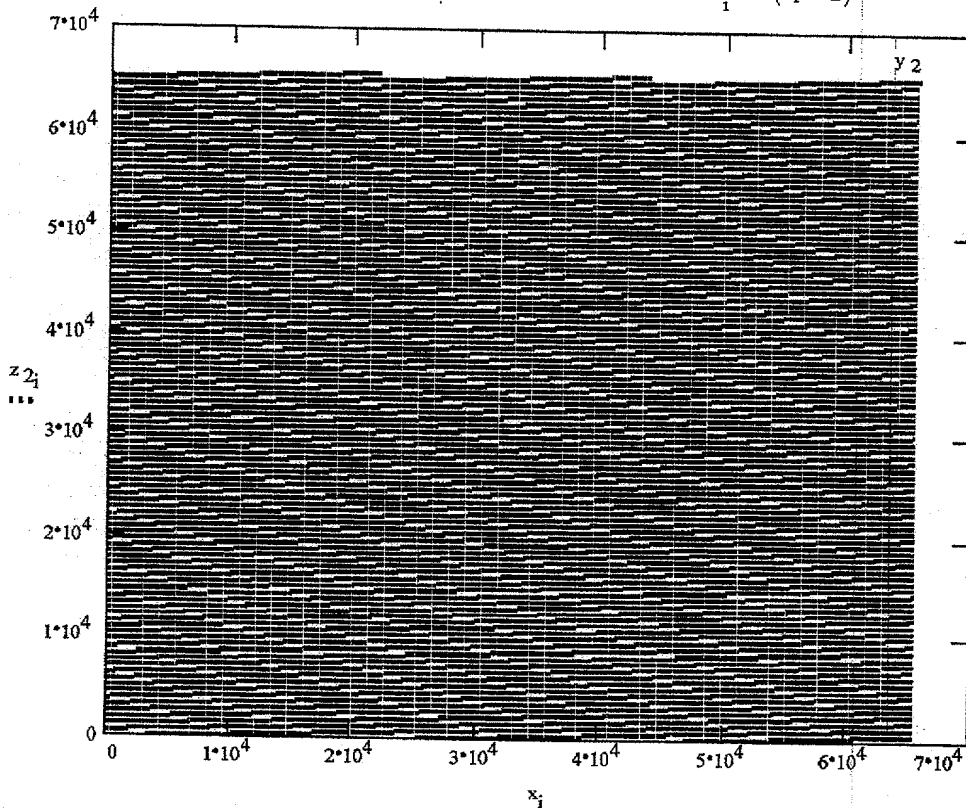
$$= x \text{ where } x < 2^N - k. \text{ (y always } < 2^N - k.)$$

$N = 16$ bits $2^N = 65536$ $k = 15$ $p = 65521$ prime modulus

pseudogroup operation with key y_1 : $y_1 = 59624$ $z_{1_i} := f(x_i, y_1)$

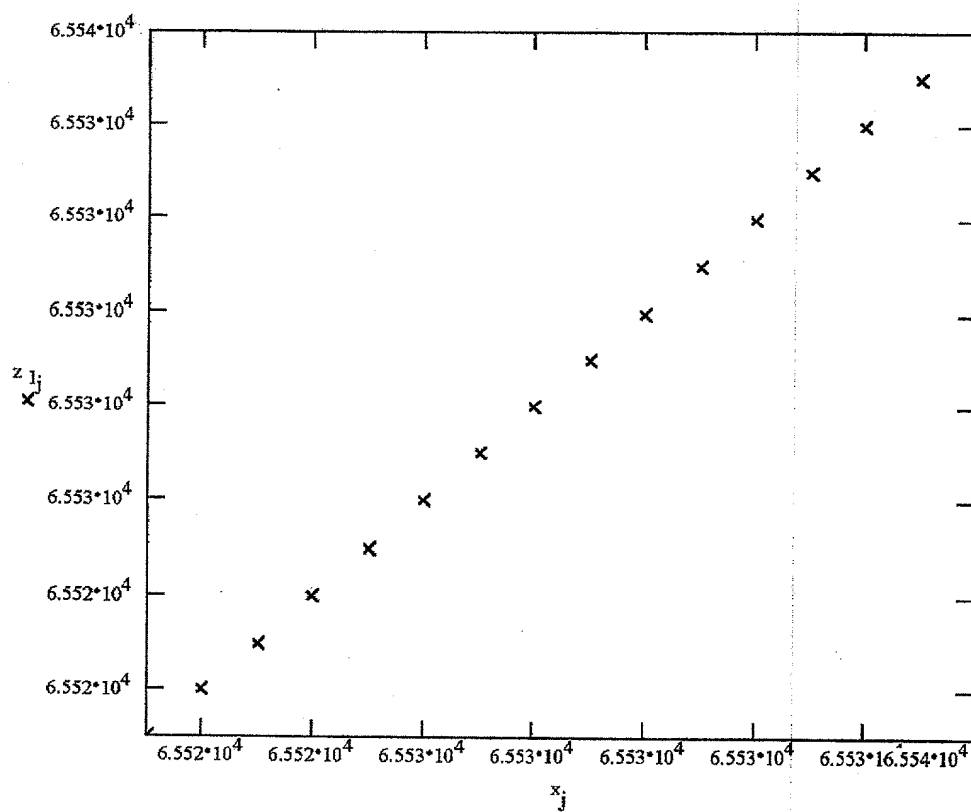
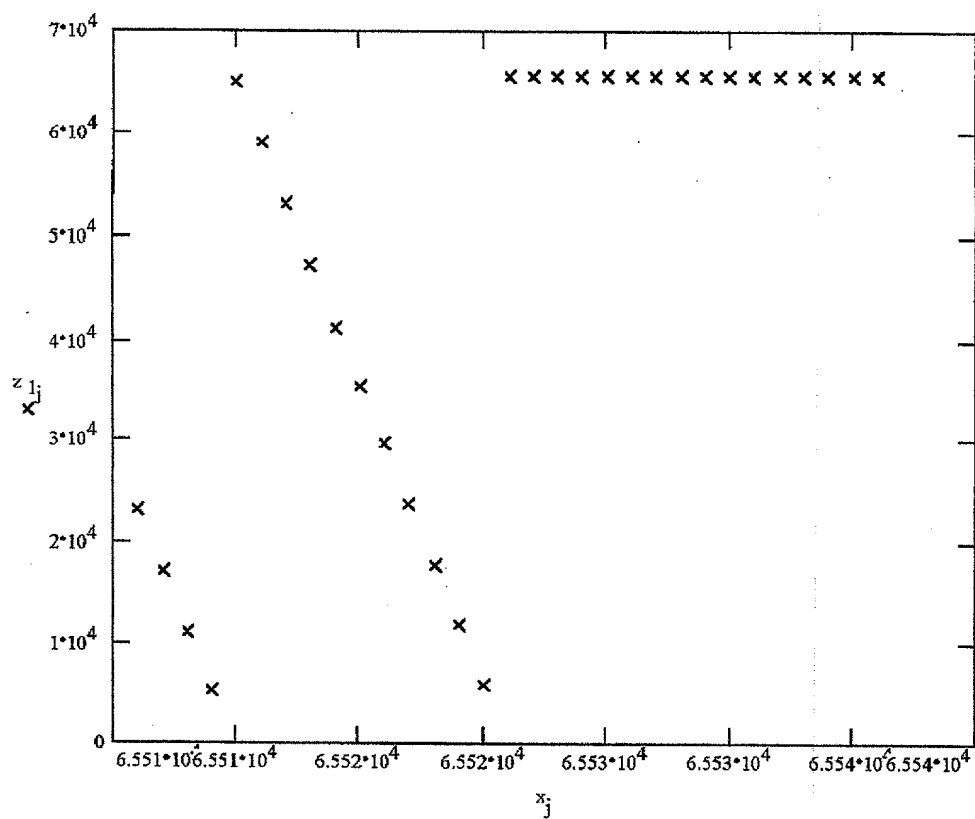


pseudogroup operation with key y_2 : $y_2 = 63631$ $z_{2_i} := f(x_i, y_2)$



[illegible]

90



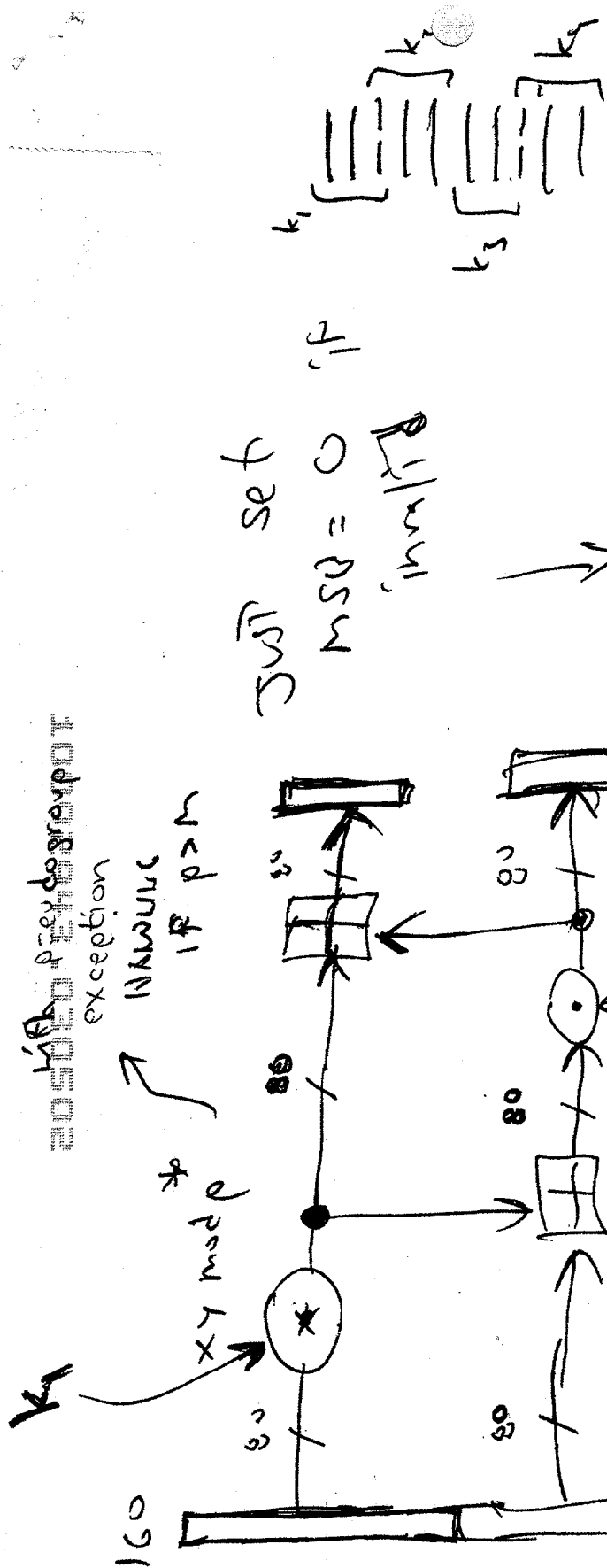
A post to sci.crypt suggested that increasing the IDEA subblock to 32-bit subblocks from the design of 16 bits would increase the security of the IDEA algorithm to a factor of 2^{32} . Lai answered that the strength of the algorithm was based on the fact that $2^{16} + 1$ is a prime, whereas $2^{32} + 1$ is not. Lai suggests that the stronger properties of the algorithm would be compromised. The point is that small changes in structure can have adverse ripple effects on the cryptographic structure that can become serious implementation errors. We look at other implementation errors in Chapter 13.

Not so limited
with
pseudogroup

$xy \bmod 2^N \pm k$

modification

to \odot



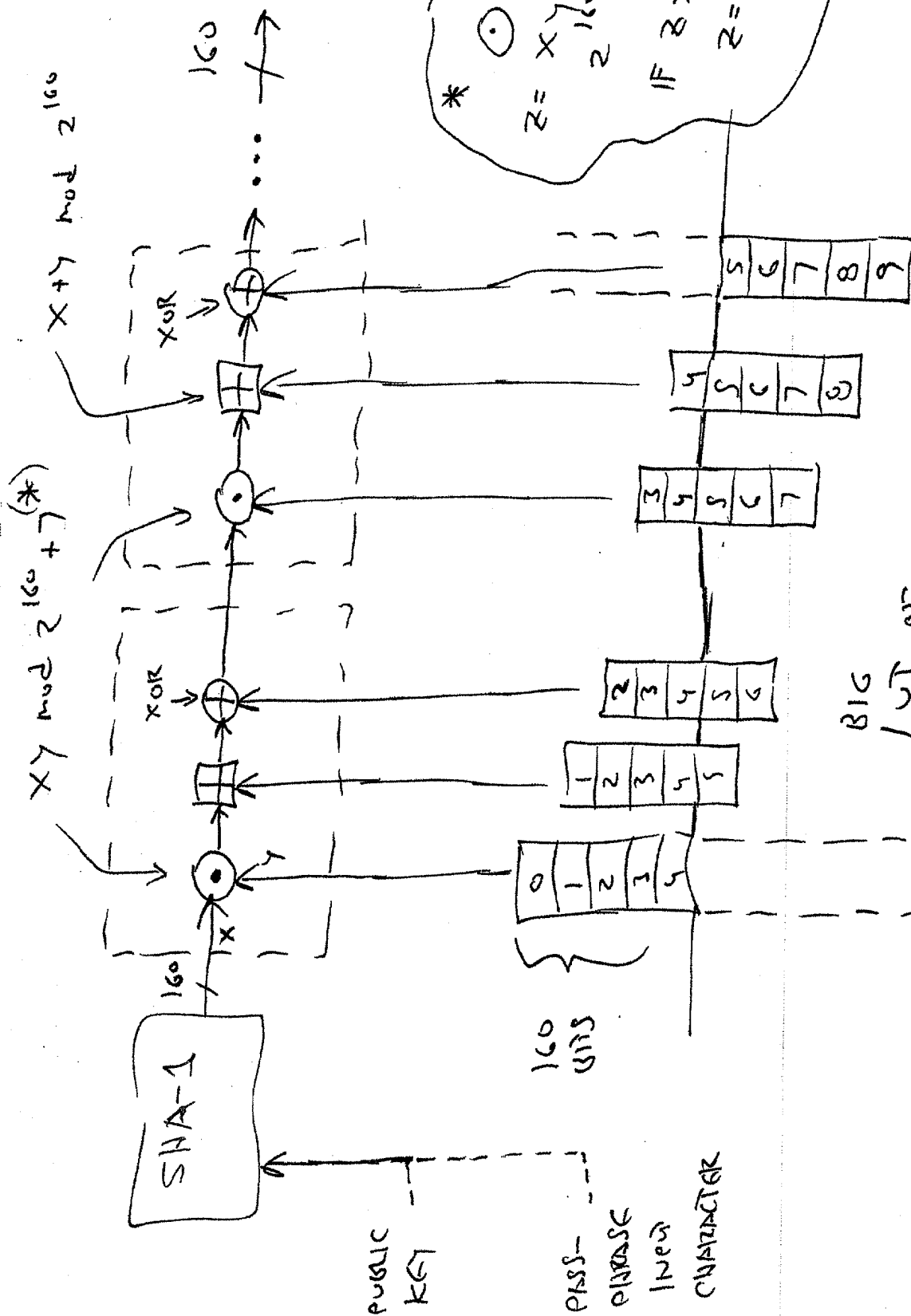
Just set
MSB = 0 if
'invalid'

$\frac{GS}{2^{80}} = 1$ of every
 1.9×10^{22} keys
is 'invalid'

$$(p - 64 < k_{ey} \leq 9)$$

$$p = 2^{80} - GS \text{ (prime)}$$

$$* [p = 2^{80} + 13 \text{ (prime)}]$$

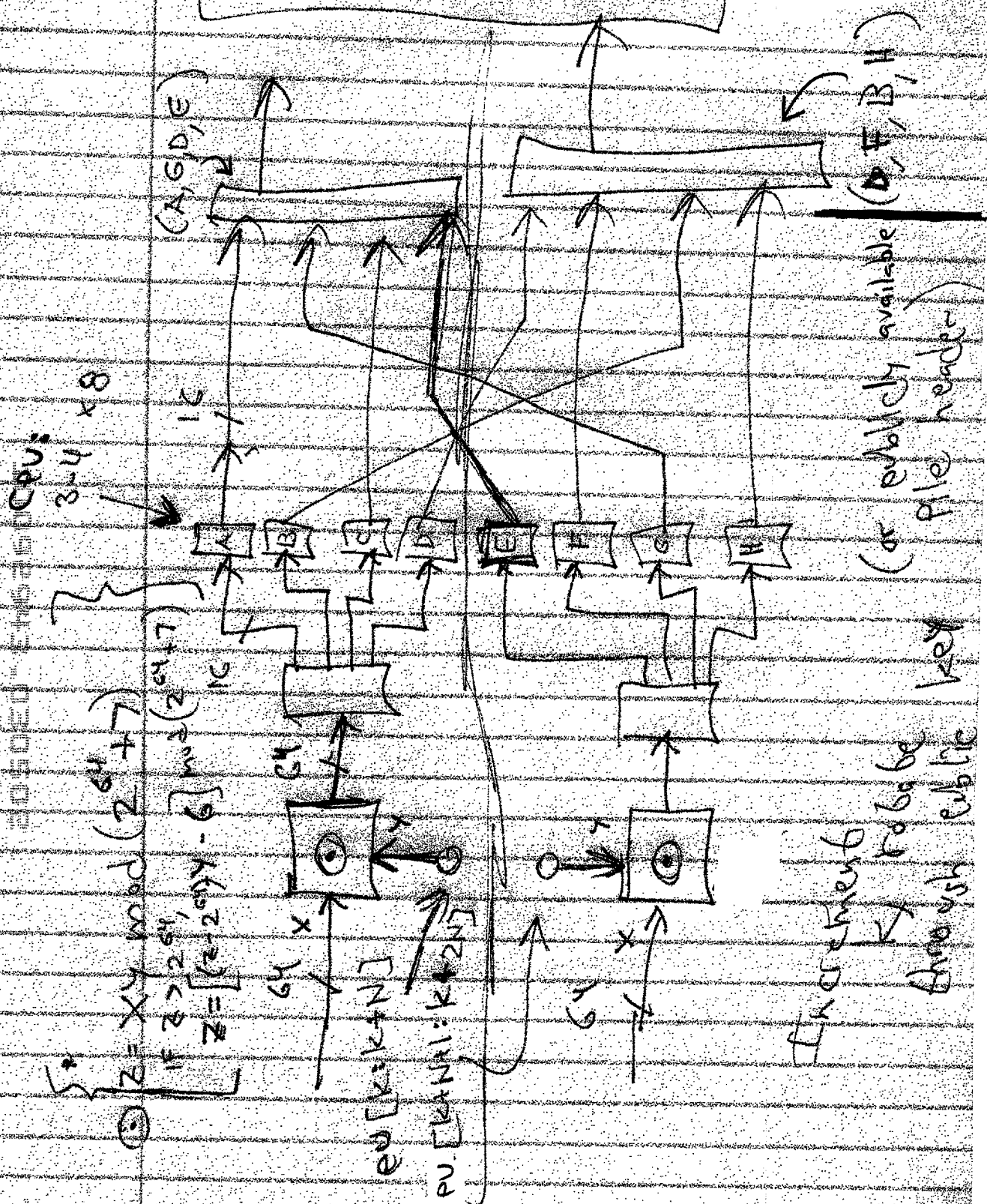
[illegible]

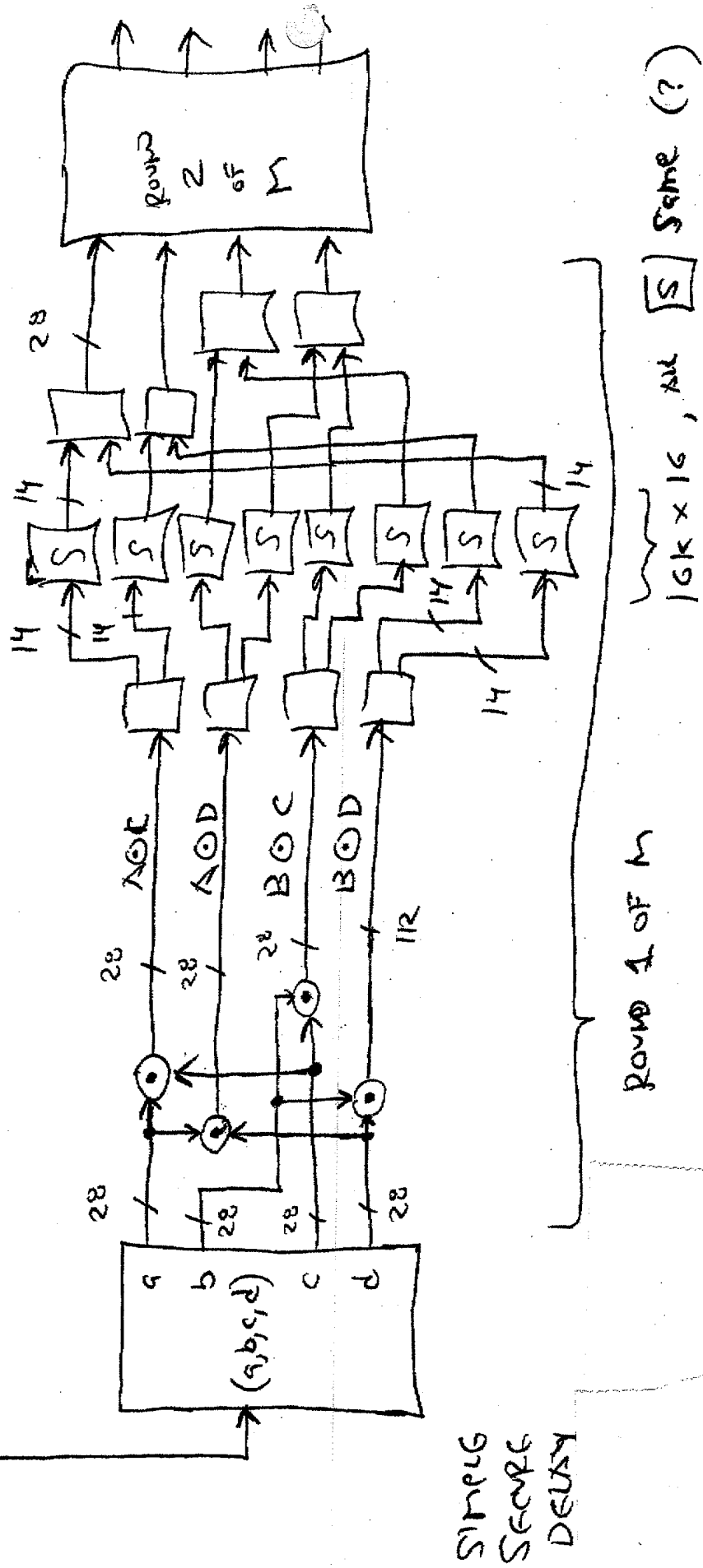
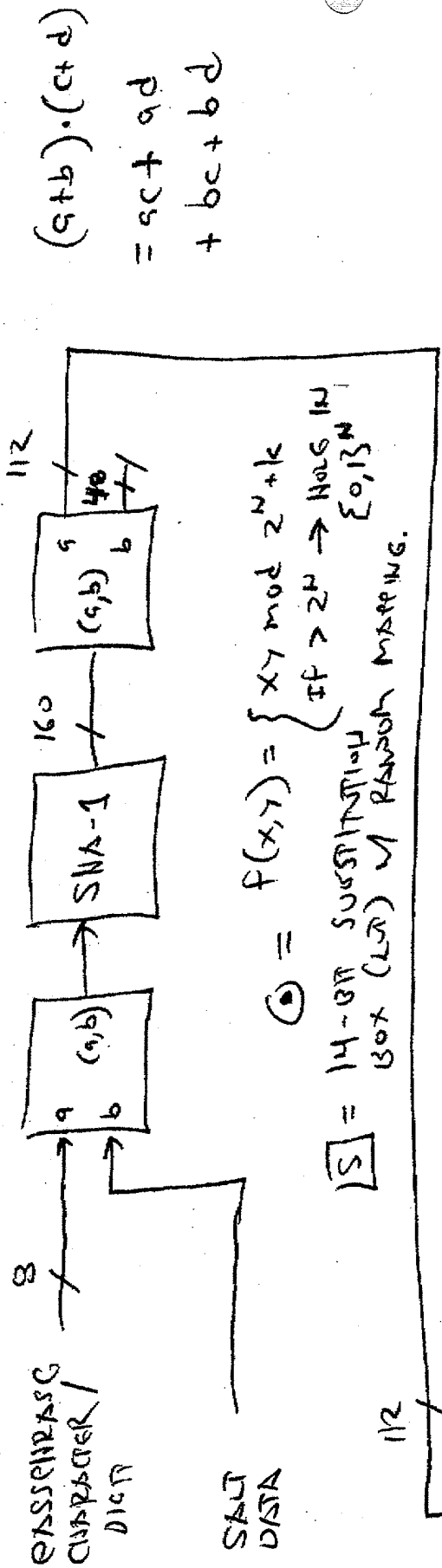
BIG
LUT OF
KNOWN BUT
RANDOMLY DISTRIBUTED
32-BIT WORDS

K-2

$\lambda - \lambda$
= same
16k x 16
LSP

NO
ITERATION
k++





[illegible][illegible]

Bit Length

Test Number (Y=prime, N=not prime, blank=not tested)																
	2^N-1	2^N-3	2^N-5	2^N-7	2^N-9	2^N-11	2^N-13	2^N-15	2^N-17							
N	N	N	N	N	N	N	N	N	N							
16	Y															
17	N	N														
18																
19	Y															
20	N	Y														
21	N	N														
22	N	Y														
23	N	N														
24	N	Y														
25	N	N														
26	N	N														
27	N	N														
28	N	N														
29	N	Y														
30	N	N														
31	Y															
32	N	N	Y													
33	N	N	N	N												
34	N	N	N	N	N											
35	N	N	N	N	N	N										
36	N	N	Y													
37	N	N	N	N	N											
38	N	N	N	N	N	N										
39	N	N	N	Y												
40	N	N	N	N	N	N	N									
41	N	N	N	N	N											
42	N	N	N	N	N	N										
43	N	N	N	N	N	N	N									
44	N	N	N	N	N	N	N									
45	N	N	N	N	N	N	N									
46	N	N	N	N	N	N	N									
47	N	N	N	N	N	N	N									
48	N	N	N	N	N	N	N									

Effective bit length for one "leaked" bit, per offset below 2^N																
2^N	1	3	5	7	9	11	13	15	17							
65536																
131072	16.0							12.0								
262144																
524288	18.0															
1048576		18.0														
2097152																
4194304		20.0														
8388608																
16777216		22.0														
33554432																
67108864																
134217728																
268435456																
536870912		27.0														
1073741824																
2147483648	30.0															
4294967296			29.4													
8589934592																
17179869184																
34359738368																
68719476736			33.4													
1.37439E+11																
2.74878E+11																
5.49756E+11				36.0												
1.09951E+12																
2.19902E+12																
4.39805E+12																
8.79609E+12																
1.75922E+13																
3.51844E+13																
7.03687E+13																
1.40737E+14																
2.81475E+14																

TABLE I-1: "Pseudogroup" Operation: $p=11$ (prime), $m=3$ bits

— KEY VALUES →

INPUT VALUES ↓

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	4	6	8	10	1	3	5
3	3	6	9	1	4	7	10	2
4	4	8	1	5	9	2	6	10
5	5	10	4	9	3	8	2	7
6	6	1	7	2	8	3	9	4
7	7	3	10	6	2	9	5	1
8	8	5	2	10	7	4	1	9

TABLE I-2: Key values AND holes associated with them

Each row contains input values producing a given output, except for holes (black squares).

Each column contains key values producing outputs for a given input, except for holes.

— KEY VALUES →

Output VALUES ↓

	1	2	3	4	5	6	7	8
1	1	6	4	3		2	8	7
2	2	1	8	6	7	4	5	3
3	3	7	1		5	6	2	
4	4	2	5	1	3	8		6
5	5	8		4	1		7	2
6	6	3	2	7		1	4	
7	7		6		8	3	1	5
8	8	4		2	6	5		1

A "hole" is an output value that will not occur for any in the set $\{1, 2, \dots, 2^m\}$ of possible key values, given a particular input value in that set.

Note from table I-1 and I-2 that the output value of 7 does not occur with Key value of 2.

TABLE II-1: (PRIOR ART) Multiplicative Group Operation $x*y \bmod p$, $p=17$ (prime), $m=4$ bits

— KEY VALUES —→

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2	4	6	8	10	12	14	16	1	3	5	7	9	11	13	15
3	3	6	9	12	15	1	4	7	10	13	16	2	5	8	11	14
4	4	8	12	16	3	7	11	15	2	6	10	14	1	5	9	13
5	5	10	15	3	8	13	1	6	11	16	4	9	14	2	7	12
6	6	12	1	7	13	2	8	14	3	9	15	4	10	16	5	11
7	7	14	4	11	1	8	15	5	12	2	9	16	6	13	3	10
8	8	16	7	15	6	14	5	13	4	12	3	11	2	10	1	9
9	9	1	10	2	11	3	12	4	13	5	14	6	15	7	16	8
10	10	3	13	6	16	9	2	12	5	15	8	1	11	4	14	7
11	11	5	16	10	4	15	9	3	14	8	2	13	7	1	12	6
12	12	7	2	14	9	4	16	11	6	1	13	8	3	15	10	5
13	13	9	5	1	14	10	6	2	15	11	7	3	16	12	8	4
14	14	11	8	5	2	16	13	10	7	4	1	15	12	9	6	3
15	15	13	11	9	7	5	3	1	16	14	12	10	8	6	4	2
16	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

INERT
VALUES
↓

205050 24626001

Unmodified Modulo

TABLE II-2: Product Operation: $p=19$ (prime), $m=4$ bits

— KEY VALUES →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2	4	6	8	10	12	14	16	18	1	3	5	7	9	11	13
3	3	6	9	12	15	18	2	5	8	11	14	17	1	4	7	10
4	4	8	12	16	1	5	9	13	17	2	6	10	14	18	3	7
5	5	10	15	1	6	11	16	2	7	12	17	3	8	13	18	4
6	6	12	18	5	11	17	4	10	16	3	9	15	2	8	14	1
7	7	14	2	9	16	4	11	18	6	13	1	8	15	3	10	17
8	8	16	5	13	2	10	18	7	15	4	12	1	9	17	6	14
9	9	18	8	17	7	16	6	15	5	14	4	13	3	12	2	11
10	10	1	11	2	12	3	13	4	14	5	15	6	16	7	17	8
11	11	3	14	6	17	9	1	12	4	15	7	18	10	2	13	5
12	12	5	17	10	3	15	8	1	13	6	18	11	4	16	9	2
13	13	7	1	14	8	2	15	9	3	16	10	4	17	11	5	18
14	14	9	4	18	13	8	3	17	12	7	2	16	11	6	1	15
15	15	11	7	3	18	14	10	6	2	17	13	9	5	1	16	12
16	16	13	10	7	4	1	17	14	11	8	5	2	18	15	12	9

INPUT
VALUES



TABLE II-3: Key values and holes associated with them

Each row contains input values producing a given output, except for holes (black squares).
Each column contains key values producing outputs for a given input, except for holes.

— KEY VALUES →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	10	13	5	4	16	11	12		2	7	8	3	15	14	6
2	2	1	7	10	8	13	3	5	15	4	14	16	6	11	9	12
3	3	11	1	15	12	10	14		13	6	2	5	9	7	4	
4	4	2	14	1	16	7	6	10	11	8	9	13	12	3		5
5	5	12	8	6	1	4		3	9	10	16	2	15		13	11
6	6	3	2	11	5	1	9	15	7	12	4	10		14	8	
7	7	13	15	16	9		1	8	5	14	11		2	10	3	4
8	8	4	9	2	13	14	12	1	3	16		7	5	6		10
9	9	14	3	7		11	4	13	1		6	15	8	2	12	16
10	10	5	16	12	2	8	15	6		1	13	4	11		7	3
11	11	15	10		6	5	7		16	3	1	12	14	13	2	9
12	12	6	4	3	10	2		11	14	5	8	1		9	16	15
13	13	16		8	14		10	4	12	7	15	9	1	5	11	2
14	14	7	11	13		15	2	16	10	9	3		4	1	6	8
15	15		5		3	12	13	9	8	11	10	6	7	16	1	14
16	16	8		4	7	9	5	2	6	13		14	10	12	15	1

OUTPUT
VALUES



A "hole" is an output value that will not occur for any in the set $\{1, 2, \dots, 2^m\}$ of possible input values, given a particular Key value in that set.

input value

Unmodified Modulo

TABLE III-1: Product Operation: $p=37$ (prime), $m=5$ bits

— KEY VALUE →

1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	1	3	5	7	9	11	13	15	17	19	21	23	25	27
3	3	6	9	12	15	18	21	24	27	30	33	36	2	5	8	11	14	17	20	23	26	29	32	35	1	4	7	10	13	16	19	22
4	4	8	12	16	20	24	28	32	36	3	7	11	15	19	23	27	31	35	2	6	10	14	18	22	26	30	34	1	5	9	13	17
5	5	10	15	20	25	30	35	3	8	13	18	23	28	33	1	6	11	16	21	26	31	36	4	9	14	19	24	29	34	2	7	12
6	6	12	18	24	30	36	5	11	17	23	29	35	4	10	16	22	28	34	3	9	15	21	27	33	2	8	14	20	26	32	1	7
7	7	14	21	28	35	5	12	19	26	33	3	10	17	24	31	1	8	15	22	29	36	6	13	20	27	34	4	11	18	25	32	2
8	8	16	24	32	3	11	19	27	35	7	16	25	34	6	15	24	33	5	14	23	32	4	13	22	31	3	12	21	30	2	11	20
9	9	18	27	36	8	17	26	35	7	16	25	34	6	15	24	33	5	14	23	32	4	13	22	31	3	12	21	30	2	11	20	29
10	10	20	30	3	13	23	33	6	16	26	36	9	19	29	2	12	22	32	5	15	25	35	8	18	28	1	11	21	31	4	14	24
11	11	22	33	7	18	29	3	14	25	36	10	21	32	6	17	28	2	13	24	35	9	20	31	5	16	27	1	12	23	34	8	19
12	12	24	36	11	23	35	10	22	34	9	21	33	8	20	34	11	25	2	16	30	7	21	35	12	26	3	17	31	8	22	36	13
13	13	26	2	15	28	4	17	30	6	19	32	8	21	34	10	23	36	12	25	1	14	27	3	16	29	4	16	28	3	15	27	2
14	14	28	5	19	33	10	24	1	15	29	6	20	34	11	25	3	18	33	11	26	4	19	34	12	27	5	20	35	13	28	6	21
15	15	30	8	23	1	16	31	9	24	2	17	32	10	25	3	18	34	13	29	8	24	3	19	35	14	30	9	25	4	20	36	15
16	16	32	11	27	6	22	1	17	33	12	28	7	23	2	18	34	13	30	10	27	7	24	4	21	1	18	35	15	32	12	29	9
17	17	34	14	31	11	28	8	25	5	22	2	19	36	16	33	13	30	11	28	9	28	10	29	11	30	12	31	13	32	14	33	16
18	18	36	17	35	16	34	15	33	14	32	13	31	12	30	11	29	10	28	9	27	8	26	7	25	6	24	5	23	4	22	3	21
19	19	1	20	2	21	3	22	4	23	5	24	6	25	7	26	8	27	9	28	10	30	13	33	16	36	19	2	22	5	25	8	28
20	20	3	23	6	26	9	29	12	32	15	35	18	1	21	4	24	7	27	10	30	13	33	16	36	19	2	22	5	25	8	28	11
21	21	5	26	10	31	15	36	20	4	25	9	30	14	35	19	3	24	8	29	13	34	18	2	23	7	28	12	33	17	1	22	6
22	22	7	29	14	36	21	6	28	13	35	20	5	27	12	34	19	4	26	11	33	18	3	25	10	32	17	2	24	9	31	16	1
23	23	9	32	18	4	27	13	36	22	8	31	17	3	26	12	35	21	7	30	16	2	25	11	34	20	6	29	15	1	24	10	33
24	24	11	35	22	9	33	20	7	31	18	5	29	16	3	27	14	1	25	12	36	23	10	34	21	8	32	19	6	30	17	4	28
25	25	13	1	26	14	2	27	15	3	28	16	4	29	17	5	30	18	6	31	19	7	32	20	8	33	21	9	34	22	10	35	23
26	26	15	4	30	19	8	34	23	12	1	27	16	5	31	20	9	35	24	13	2	28	17	6	32	21	10	36	25	14	3	29	18
27	27	17	7	34	24	14	4	31	21	11	1	28	18	8	35	25	15	5	32	22	12	2	29	19	9	36	26	16	6	33	23	13
28	28	19	10	1	29	20	11	2	30	21	12	3	31	22	13	4	32	23	14	5	33	24	15	6	34	25	16	7	35	26	17	8
29	29	21	13	5	34	26	18	10	2	31	23	15	7	36	28	20	12	4	33	25	17	9	1	30	22	14	6	35	27	19	11	3
30	30	23	16	9	2	32	25	18	11	4	34	27	20	13	6	36	29	22	15	8	1	31	24	17	10	3	33	26	19	12	5	35
31	31	25	19	13	7	1	32	26	20	14	8	2	33	27	21	15	9	3	34	28	22	16	10	4	35	29	23	17	11	5	36	30
32	32	27	22	17	12	7	2	34	29	24	19	14	9	4	36	31	26	21	16	11	6	1	33	28	23	18	13	8	3	35	30	25

Overflowing values (>32), which would be remapped to holes with pseudogroup operation, are shown in italics.

Example of holes:

29, 31 do not occur given $K=2$

INPUT
VALUES

TABLE III-2: Key values and holes associated with them
Each row contains keys values producing a given output, except for holes (black squares).
Each column contains key values producing outputs for a given input, except for holes.

KEY VALUES →

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	19	25	28	15	31	16	14	26	27	20	8	5	7	24	2	13	30	32	29	17	3	10	11	4	23	21	6	22			
2	1	13	19	30	25	32	28	29	15	17	31	3	16	10	14	11	4	26	23	27	21	6	20	22	8	9	5	12	7		
3	20	1	10	8	19	11	5	25	4	7	28	23	24	15	21	31	6	2	16	22	13	14	9	30	12	32	26	18	29		
4	2	26	1	23	13	27	19	21	30	25	6	32	20	28	22	29	8	15	9	17	5	31	12	3	7	16	18	10	24	14	
5	21	14	29	1	7	6	17	19	24	22	26	3	25	9	27	10	28	2	12	11	15	13	18	20	4	31	30				
6	3	2	20	16	1	22	10	13	8	14	19	9	11	30	5	25	12	4	32	7	26	28	18	23	29	24	27	15	5	6	
7	7	22	27	11	31	32	1	24	9	4	16	29	19	12	20	23	14	17	25	2	18	8	21	6	14	32	20	11	28		
8	4	15	2	9	26	17	1	5	23	31	13	12	27	3	19	7	21	16	30	18	10	25	24	6	14	32	20	11	28		
9	23	3	30	24	20	15	1	12	21	10	32	8	26	31	19	18	6	11	29	2	5	27	16	25	3	8	25	23			
10	5	28	21	2	14	12	29	1	11	7	15	6	13	18	17	20	19	4	24	31	22	30	26	10	7	31	9	29	20		
11	24	16	12	17	8	28	6	30	27	1	4	14	18	3	5	15	22	32	19	23	2	19	32	15	3	14	4	27			
12	6	4	3	32	2	7	20	26	16	28	1	18	22	23	10	29	13	24	8	27	14	15	19	9	21	11	17	30	5		
13	25	29	31	10	23	2	11	18	31	8	32	21	1	24	3	9	28	13	4	16	5	29	6	19	26	10	12				
14	7	17	22	25	27	2	11	18	31	8	32	21	1	24	3	9	28	13	4	16	5	29	6	19	26	10	12				
15	26	5	13	3	21	18	25	14	20	9	25	26	4	9	1	31	27	7	30	10	6	28	8	2	17	23	12	19	16		
16	8	30	4	18	15	2	10	9	25	26	24	17	6	1	14	5	32	23	31	20	13	11	12	28	27	3	22	19			
17	27	18	32	9	13	16	6	15	23	7	25	11	8	1	3	1	12	22	21	4	10	17	32	13	7	8	26				
18	9	6	23	11	3	29	30	2	24	5	20	27	16	15	25	1	12	22	21	4	10	17	32	13	7	8	26				
19	28	31	14	26	8	7	13	32	17	10	4	21	22	12	1	25	15	16	27	20	5	24	2	30	29	3	11				
20	10	19	5	4	28	24	21	31	2	22	14	30	12	26	29	3	1	8	11	25	7	23	15	6	16	13	9				
21	29	7	19	22	3	27	28	12	11	13	20	31	23	32	5	14	1	6	17	24	26	25	9	10	2	15	18				
22	11	32	24	16	19	12	23	17	2	8	28	6	10	30	7	27	31	1	9	4	29	20	14	25	18	21	3				
23	30	20	15	12	10	26	19	6	29	5	16	4	13	28	9	3	24	1	21	32	8	31	18	11	2	27	25				
24	12	8	6	27	4	14	3	15	32	19	2	7	9	20	21	26	11	16	17	28	30	1	18	5	22	23	10				
25	31	25	20	5	30	17	11	21	9	19	15	14	27	8	24	13	29	10	23	22	18	1	28	16	26	20	7	2	32		
26	13	21	25	20	29	9	31	7	10	26	30	22	31	24	4	19	20	17	18	13	6	15	7	11	1	29	12	14	2		
27	32	9	16	23	25	8	3	26	16	27	5	2	29	11	6	18	19	31	26	8	32	10	21	12	1	15	20	24			
28	14	22	28	11	20	32	14	6	24	25	10	18	30	16	21	7	19	3	27	12	13	31	23	5	1	17	26	9			
29	22	10	26	6	5	13	28	3	21	8	18	2	25	17	14	23	20	12	19	29	16	4	9	24	1	32	31				
30	15	10	26	6	5	13	28	3	21	8	18	2	25	17	14	23	20	12	19	29	16	4	9	24	1	32	31				
31	17	21	15	27	24	29	23	18	28	26	7	32	4	12	25	5	30	11	9	19	14	8	13	10	22	1	16				
32	16	23	8	30	31	4	20	18	13	15	11	12	2	28	10	27	9	25	3	26	22	24	19	17	6	7	1				

A "hole" is an output value that will not occur for any in the set $\{1, 2, \dots, 2^m\}$ of possible input values, given a particular key value in that set.

input value

Encryption (forward)

$a = \text{plaintext}$

$x = \text{key}$

1. $z = ax \bmod p$

2. If $z > M$,

2.1 $k = z - M$

2.2 $z = kx - (p - M - 1) \bmod p$

end

Sub-block size



$$M = 2^N$$

$p = \text{prime just above } M$

Decryption (inverse)

$z = \text{ciphertext}$

$x^{-1} =$

inverse key

1. $k = x^{-1} (z + (p - M - 1)) \bmod p$

2. If $1 \leq k \leq (p - M - 1)$, ^{2.1} $z = M + k$; end

2.2 $a = zx^{-1} \bmod p$

↑
VALUES
HOLD
OUT

✓

11)

$$\therefore \gamma_k = k \cdot x - (p - (k + 1))$$

2

P-1

32- 31- 30- 29- 28- 27- 26- 25- 24- 23- 22- 21- 20- 19- 18- 17- 16- 15- 14- 13- 12- 11- 10- 9- 8- 7- 6- 5- 4- 3- 2- 1-

$\begin{matrix} 5 & 5 \\ 6 & 6 \\ 11 & 11 \\ 2 & 3 \end{matrix}$

$$y_2 = 2 \cdot (x-1) = 2x-2$$

$$k_1 = 1 \quad k_2 = 2$$

4-1

P-2

[illegible]

$$y_8(x=3) = 16$$

$$1/2 = 2 \cdot x - 8$$

(3,16) should be a hole - it is!

$$y_3 = 3x - 8$$

$$L = 1.25 \times 10^6$$

55

56115

$$73 - 65 = 8$$

$$F_{\alpha\beta} \quad x = 51, k = 7$$

$$y = 7.51 - 8 \ln 273$$

$$= Gg(ho)$$

```

function h = holes(y,p,M,short);

% h = holes(y,p,N, short);
% CONFIDENTIAL AND PROPRIETARY
% Edwin A. Suominen

% Finds "holes" - skipped values of set  $\{0,1\}^N$  in result
% of  $x*y \bmod p$ .

% Number of values in set  $S:\{0,1\}^N$ 
%  $M = 2^N$ ;

% For vector inputs...
for k=1:length(y)

    s = 1:M;    % Working array of values in set S

    % Zero out values in set that occur ("non-holes")
    for i = 1:M
        j = product(i,y(k),p); %  $xy \bmod p$ 
        % Zero out if not a hole
        if j<=M, s(j) = 0; end
    end

    % Sort decending to get holes first
    z = -sort(-s);

    if nargin>3,
        % There can be no more than  $p-2^N-1$  holes.
        % Limit size of result vector(s) accordingly.
        z = z(1:p-M-1);
    % Add result vector to array (if vector y)
    h(:,k) = z';
    else
        h(:,k) = s';
    end

end

end

```

205509-010500T

```

function [y1,y2] = holeplot(p,M);

% y = holeplot(p,M)

% Modulus is 2^N+k, where k is odd
% p = M + k;

% Create 1010...matrix of holes
for i = 1:M/2,
    % Get holes for each column
    s = holes(M+1-i,p,M);
    % Convert to 1010... format
    s = s>0;
    % Convert to text string
    col = M+1-i
    if col<10,
        r1 = [' ' num2str(col,2)];
    else
        r1 = num2str(col,2);
    end
    y1(i,:) = [ r1 '-' num2str(s,1)];
end

for i = 1:M/2,
    % Get holes for each column
    s = holes(M+1-(i+M/2),p,M);
    % Convert to 1010... format
    s = s>0;
    % Convert to text string
    col = M/2+1-i
    if col<10,
        r1 = [' ' num2str(col,2)];
    else
        r1 = num2str(col,2);
    end
    y2(i,:) = [ r1 '-' num2str(s,1)];
end

```

2050ED E462600F

HOLES.M

```
## usage: h = holes(y,p,M)
## Function discovered by Edwin A. Suominen
## Written for Octave (GNU MATLAB alternative)
```

```
function [h,yi] = holes(y,p,M)
```

```
h = zeros(p-M-1,2);
```

```
## Compute inverse of y mod p
```

```
[d,yi]= gcd(p,y);
```

```
if ( yi(2) < 0 )
```

```
    yi = p + yi(2);
```

```
elseif
```

```
    yi = yi(2);
```

```
endif
```

```
## Compute column 1 of LUT for this key y:
```

```
## holes in ascending order
```

```
kk = 0; # Counter for iterating next valid hole value
```

```
## For all possible hole values...
```

```
for i = 1:p-M-1
```

```
    ## Compute prospective hole value (may not be valid)
```

```
    h(i,1) = M+1 - rem( i*y-(p-M-1) ,p);
```

```
    ## If not valid (if >M), set to flag value
```

```
    if ( (h(i,1)>M) | (h(i,1)<1) )
```

```
        h(i,:) = zeros(1,2);
```

```
    else
```

```
        kk++; # Increment valid holes counter
```

```
    endif
```

```
endfor
```

```
## Compute column 2 of LUT for this key y: all possible
```

```
## overflowing values, xy mod p > M, in ascending order
```

```
kk2 = 1; # Counter for iterating next poss. overflow value
```

```
## For M+1 (lowest overflow) to p-1 (highest possible)...
```

```
for i = M+1:p-1
```

```
    ## Compute input value that would produce each
```

```
    ## possible overflowing output
```

```
    if ( rem(i*yi,p) <= M ) # If input valid...
```

```
        h(kk2,2) = i; # ...assign overflow to LUT.
```

```
        kk2++; # Move to next available LUT entry
```

```
    endif
```

```
endfor
```

```
## Sort ascending by values in each column
```

```
h = sort(h);
```

```
if ( 1+(length(h)-kk) > length(h) )
```

```
h = 0; # If there are no holes (e.g., for y=1)
else
h = h(1+(length(h)-kk):length(h),:); # Shrink h to omit flag values
endif

endfunction
```

205050.84625007

ENCRYPT.M

```
## usage: z = encrypt (x,y,N,k)
## INPUT: input block(s) x, key y, block length N in bits,
## k offset of modulus from 2^N (p=2^N+k)
## OUTPUT: z = x*y mod (2^N+k), but if
## z >= p, z = ( (z-2^N)*y - (k-1) ) mod (2^N+k )
## Function discovered by Edwin A. Suominen
## Written for Octave (GNU MATLAB alternative)

function z = encrypt(x,y,N,k)

L = length(x); # multiple input blocks can be supplied in a vector
z = zeros(L,1); # initialize output vector

## Enforce k must be odd
if ((k/2)==floor(k/2))
    disp('2^N+k cannot be prime is k is even!');
    return;
endif

## Define set order (M) and modulus (p)
M = 2^N; p = M+k;

## Compute LUT of holes in ascending order
## for this key y
h = holes(y,p,M);
Nh = rows(h);

## Basic modulo multiplication operation
## Do as array to speed things up
z = rem(x.*y,p);

## For each element in vector...
for i = 1:L

    ## Inventive exception handling
    if (z(i) > M)
        ## Map overflowing value to corresponding hole value in LUT
        ## If there are no holes (h=1 scalar), this code will not
        ## be called because z will always be <= M.
        c = 1:Nh; c = c'; # 1,2 ... (# of valid holes)
        c = c .* ( (z(i)*ones(Nh,1))==h(:,2) ); # Zeros with index of match
        ## z = hole from LUT entry having matching overflow value
        z(i) = h(max(c),1);
    endif

endfor
endfunction
```

DECRYPT.M

```
## usage: x = decrypt (z,y,N,k)
## INPUT: encoded block(s) z, key y, block length N in bits,
## k offset of modulus from 2^N (p=2^N+k)
## OUTPUT: x = z*y^-1 mod (2^N+k), but if
## z = h, where h = ( ((1:k)*y - (k-1) ) mod (2^N+k) )
## then z = M+a, where
## a = y^-1 * (2*M+(2+p-h)) mod (2^N+k)
## Function discovered by Edwin A. Suominen
## Written for Octave (GNU MATLAB alternative)

function x = decrypt (z,y,N,k)

L = length(z); # multiple input blocks can be supplied in a vector

## Enforce k must be odd
if ((k/2)==floor(k/2))
    disp('2^N+k cannot be prime is k is even!');
    return;
endif

## Define set order (M) and modulus (p)
M = 2^N; p = M+k;

## Compute LUT of holes in ascending order
## for this key y
[h,y] = holes(y,p,M); # With two args out, returns y^-1
if ( size(h)==1 )
    Nh = 0; # Account for special case of no holes
else
    Nh = rows(h);
endif

## Done with encryption key y, now y is modulo inverse of orig. y

## For all encrypted blocks (values)...
for i = 1:L

    if Nh>0
        ## If z(i) has been mapped to a hole, restore to overflowing value
        ## For all possible hole values given this key
        for j = 1:Nh
            ## If matches a hole value, remap back
            if (z(i)==h(j,1)), z(i) = h(j,2); endif
        endfor
    endif

endfor
```

```
## Now invert remapped values in vector
## Restored overflowing values will be decrypted properly.
## Do as array to speed things up
x = rem(z.*y,p);      # y = y^1 at this point

endfunction
```

2050E0-E462500F

HOLETEST.M

HOLETEST.M
Written for Octave (GNU MATLAB alternative)

Np = 1; M = 128; p = M + 3;
Np = 1; M = 512; p = M + 9;

% Try all column (key) values in {1,2,...M}
for j = 2:M,

% Get hole values with brute-force lookup method
x1 = holes1(j,p,M);

% Get hole values using formula discovered by Ed Suominen
x2 = holes2(j,p,M);

disp('');
disp(['j=',num2str(j)]);
disp(' -x1- -x2-');
disp([x1 x2]);

% Compare
err(j) = sum(abs(x1-x2));
disp(['Sum of absolute differences = ',num2str(err(j))]);

endfor

HOLES1.M

```
function h = holes1(y,p,M);

% h = holes1(y,p,N);
% Finds "holes" - skipped values of set {0,1}^N in result
% of x*y mod p. Variable length result with only holes.

% Number of values in set S:{0,1}^N
% M = 2^N;

s = 1:M;    % Working array of values in set S

% Zero out values in set that occur ("non-holes")
for i = 1:M
    j = rem(i*y,p); % xy mod p
    % Zero out if not a hole
    if j<=M, s(j) = 0; end
endfor

% Sort decending to get holes first
h = -sort(-s);
% Trim off zeros (non-holes)
Nnz = sum(h>0); h = h(1:Nnz)';

endfunction
```

205050-030505

HOLES2.M

```
function h = holes2(y,p,M);

% h = holes2(y,p,M)
% Finds "holes" - skipped values of set  $\{0,1\}^N$  in result
% of  $x*y \bmod p$ .
% Uses equation discovered by Edwin A. Suominen

% Number of values in set  $S:\{0,1\}^N$ 
%  $M = 2^N$ ;

k = p-(M+1);

% For vector inputs...
for i=1:length(y)
    for j=1:k,
        ## Input values between M+1 and p will of necessity
        ## be mapped to holes (values not produced by inputs
        ## from set  $\{1,2,\dots,M\}$  because  $xy \bmod p$  is a bijection
        ## (See HAC 1.8 Definition)
        ##  $h(j,i) = \text{rem}((M+j)*y, p)$ ;
        ## Equation above is simple but doesn't work when
        ##  $M < xy < p$  (which happens rarely, but it happens).

        h(j,i) = M+1 - rem(j*y(i)-k,p);
    endfor
endfor

% Map negs. to 0, Sort decending to match formats
Nok = sum(h<=M); h = sort(h); h = h(1:Nok);
if Nok==0, h = []; endif
h = h.*(h>0);
h = -sort(-h);
% Trim off zeros (non-holes)
Nnz = sum(h>0); h = h(1:Nnz);

endfunction
```

[illegible]

```
octave:56> date
ans = 
octave:57> clock
ans =
```

2000.0000	9.0000	30.0000
12.0000	35.0000	32.0890

```
octave:58> type holes1
holes1 is the user-defined function
defined from: /1093-2/holes1.m
```

```
function h = holes1(y,p,M);

% h = holes1(y,p,N);
% CONFIDENTIAL AND PROPRIETARY
% Edwin A. Suominen
% 091600 - Initial writing
% Finds "holes" - skipped values of set
% {0,1}^N in result
% of x*y mod p. Variable length result
% with only holes.
```

```
% Number of values in set S: {0,1}^N
% M = 2^N;

s = 1:M;      % Working array of values in
set S
```

```
% Zero out values in set that occur
("non-holes")
for i = 1:M
    j = rem(i*y,p); % xy mod p
    % Zero out if not a hole
    if j<=M, s(j) = 0; end
endfor
```

```
% Sort decending to get holes first
h = -sort(-s);
% Trim off zeros (non-holes)
Nnz = sum(h>0); h = h(1:Nnz)';
```

```
endfunction octave:59> type holes2
holes2 is the user-defined function
defined from: /1093-2/holes2.m
```

```
function h = holes2(y,p,M);
```

```
% h = holes2(y,p,M)
```

```
% Finds "holes" - skipped values of set
% {0,1}^N in result
% of x*y mod p.
% Uses equation discovered by EAS
```

```
% Number of values in set  $S:\{0,1\}^N$ 
%  $M = 2^N$ ;
```

$$k = p - (M+1);$$

```
% For vector inputs...
for i=1:length(y)
```

for $j=1:k$,

```

    ## Input values between M+1 and p
will of necessity
    ## be mapped to holes (values not
produced by inputs
    ## from set {1,2,...M} because  $xy \bmod p$ 
p is a bijection
    ## (See HAC 1.8 Definition)
    h(j,i) = rem( (M+j)*y , p);

```

```

      ## The simple equation above is
substituted for the one below
      ## h(j,i) = M+1 - rem(j*y(i)-k,p);
    endfor
  endfor
end

```

endfor

```
% Map negs. to 0, Sort descending to match
formats
Nok = sum(h<=M); h = sort(h); h =
h(1:Nok);
if Nok==0, h = []; endif
h = h.*(h>0);
h = -sort(-h);
% Trim off zeros (non-holes)
Nnz = sum(h>0); h = h(1:Nnz);
```

```
endfunction octave:60> type holetest
holetest is the script file: /1093-
2/holetest.m
```

```
## HOLETEST.M
## This file is CONFIDENTIAL AND
## PROPRIETARY.
```

```
## Written for Octave (GNU MATLAB
alternative)
## REVISION
```

```

Np = 1; M = 512; p = M + 9;

% Try all column (key) values in
{1,2,...M}
for j = 2:M,

% Get hole values with brute-force lookup
method
x1 = holes1(j,p,M);

% Get hole values using formula
discovered
% by Ed Suominen
x2 = holes2(j,p,M);

disp('');
disp(['j=',num2str(j)]);
disp(' -x1- -x2-');
disp([x1 x2]);

% Compare
err(j) = sum(abs(x1-x2));
disp(['Sum of absolute differences = ',num2str(err(j))]);

endfor octave:61> who

*** currently compiled functions:

clock date holes1 holes2

octave:62> holetest

j=2
-x1- -x2-
511 511
509 509
507 507
505 505
Sum of absolute differences = 0

j=3
-x1- -x2-
512 512
509 509
506 506
503 503
500 500
497 497
Sum of absolute differences = 0

j=4
-x1- -x2-
509 509

```

```

505 505
501 501
497 497
493 493
489 489
Sum of absolute differences = 0

j=5
-x1- -x2-
511 511
506 506
501 501
496 496
491 491
486 486
481 481
Sum of absolute differences = 0

j=6
-x1- -x2-
509 509
503 503
497 497
491 491
485 485
479 479
473 473
Sum of absolute differences = 0

j=7
-x1- -x2-
507 507
500 500
493 493
486 486
479 479
472 472
465 465
Sum of absolute differences = 0

j=8
-x1- -x2-
505 505
497 497
489 489
481 481
473 473
465 465
457 457
Sum of absolute differences = 0

j=9
-x1- -x2-
512 512
503 503
494 494
485 485
476 476

```

205050 E462600F

467 467
458 458
449 449
Sum of absolute differences = 0

j=10
-x1- -x2-
511 511
501 501
491 491
481 481
471 471
461 461
451 451
441 441
Sum of absolute differences = 0

j=11
-x1- -x2-
510 510
499 499
488 488
477 477
466 466
455 455
444 444
433 433
Sum of absolute differences = 0

j=12
-x1- -x2-
509 509
497 497
485 485
473 473
461 461
449 449
437 437
425 425
Sum of absolute differences = 0

j=13
-x1- -x2-
508 508
495 495
482 482
469 469
456 456
443 443
430 430
417 417
Sum of absolute differences = 0

j=14
-x1- -x2-
507 507
493 493
479 479

465 465
451 451
437 437
423 423
409 409
Sum of absolute differences = 0

j=15
-x1- -x2-
506 506
491 491
476 476
461 461
446 446
431 431
416 416
401 401
Sum of absolute differences = 0

j=16
-x1- -x2-
505 505
489 489
473 473
457 457
441 441
425 425
409 409
393 393
Sum of absolute differences = 0

j=17
-x1- -x2-
504 504
487 487
470 470
453 453
436 436
419 419
402 402
385 385
Sum of absolute differences = 0

j=18
-x1- -x2-
503 503
485 485
467 467
449 449
431 431
413 413
395 395
377 377
Sum of absolute differences = 0

j=19
-x1- -x2-
502 502

2050E0346500T

162 162
108 108
54 54
Sum of absolute differences = 0

j=468
-x1- -x2-
424 424
371 371
318 318
265 265
212 212
159 159
106 106
53 53
Sum of absolute differences = 0

j=469
-x1- -x2-
416 416
364 364
312 312
260 260
208 208
156 156
104 104
52 52
Sum of absolute differences = 0

j=470
-x1- -x2-
408 408
357 357
306 306
255 255
204 204
153 153
102 102
51 51
Sum of absolute differences = 0

j=471
-x1- -x2-
400 400
350 350
300 300
250 250
200 200
150 150
100 100
50 50
Sum of absolute differences = 0

j=472
-x1- -x2-
392 392
343 343
294 294

245 245
196 196
147 147
98 98
49 49
Sum of absolute differences = 0

j=473
-x1- -x2-
384 384
336 336
288 288
240 240
192 192
144 144
96 96
48 48
Sum of absolute differences = 0

j=474
-x1- -x2-
376 376
329 329
282 282
235 235
188 188
141 141
94 94
47 47
Sum of absolute differences = 0

j=475
-x1- -x2-
368 368
322 322
276 276
230 230
184 184
138 138
92 92
46 46
Sum of absolute differences = 0

j=476
-x1- -x2-
360 360
315 315
270 270
225 225
180 180
135 135
90 90
45 45
Sum of absolute differences = 0

j=477
-x1- -x2-
352 352

308 308
264 264
220 220
176 176
132 132
88 88
44 44
Sum of absolute differences = 0

j=478
-x1- -x2-
344 344
301 301
258 258
215 215
172 172
129 129
86 86
43 43

Sum of absolute differences = 0

j=479
-x1- -x2-
336 336
294 294
252 252
210 210
168 168
126 126
84 84
42 42

Sum of absolute differences = 0

j=480
-x1- -x2-
328 328
287 287
246 246
205 205
164 164
123 123
82 82
41 41

Sum of absolute differences = 0

j=481
-x1- -x2-
320 320
280 280
240 240
200 200
160 160
120 120
80 80
40 40

Sum of absolute differences = 0

j=482

-x1- -x2-
312 312
273 273
234 234
195 195
156 156
117 117
78 78
39 39

Sum of absolute differences = 0

j=483
-x1- -x2-
304 304
266 266
228 228
190 190
152 152
114 114
76 76
38 38

Sum of absolute differences = 0

j=484
-x1- -x2-
296 296
259 259
222 222
185 185
148 148
111 111
74 74
37 37

Sum of absolute differences = 0

j=485
-x1- -x2-
288 288
252 252
216 216
180 180
144 144
108 108
72 72
36 36

Sum of absolute differences = 0

j=486
-x1- -x2-
280 280
245 245
210 210
175 175
140 140
105 105
70 70
35 35

Sum of absolute differences = 0

2050E0 24625007

j=487

-x1-	-x2-
272	272
238	238
204	204
170	170
136	136
102	102
68	68
34	34

Sum of absolute differences = 0

j=488

-x1-	-x2-
264	264
231	231
198	198
165	165
132	132
99	99
66	66
33	33

Sum of absolute differences = 0

j=489

-x1-	-x2-
256	256
224	224
192	192
160	160
128	128
96	96
64	64
32	32

Sum of absolute differences = 0

j=490

-x1-	-x2-
248	248
217	217
186	186
155	155
124	124
93	93
62	62
31	31

Sum of absolute differences = 0

j=491

-x1-	-x2-
240	240
210	210
180	180
150	150
120	120
90	90
60	60

30 30

Sum of absolute differences = 0

j=492

-x1-	-x2-
232	232
203	203
174	174
145	145
116	116
87	87
58	58
29	29

Sum of absolute differences = 0

j=493

-x1-	-x2-
224	224
196	196
168	168
140	140
112	112
84	84
56	56
28	28

Sum of absolute differences = 0

j=494

-x1-	-x2-
216	216
189	189
162	162
135	135
108	108
81	81
54	54
27	27

Sum of absolute differences = 0

j=495

-x1-	-x2-
208	208
182	182
156	156
130	130
104	104
78	78
52	52
26	26

Sum of absolute differences = 0

j=496

-x1-	-x2-
200	200
175	175
150	150
125	125
100	100

2050E0-8462500F

75 75
50 50
25 25
Sum of absolute differences = 0

j=497
-x1- -x2-
192 192
168 168
144 144
120 120
96 96
72 72
48 48
24 24

Sum of absolute differences = 0

j=498
-x1- -x2-
184 184
161 161
138 138
115 115
92 92
69 69
46 46
23 23

Sum of absolute differences = 0

j=499
-x1- -x2-
176 176
154 154
132 132
110 110
88 88
66 66
44 44
22 22

Sum of absolute differences = 0

j=500
-x1- -x2-
168 168
147 147
126 126
105 105
84 84
63 63
42 42
21 21

Sum of absolute differences = 0

j=501
-x1- -x2-
160 160
140 140
120 120

100 100
80 80
60 60
40 40
20 20

Sum of absolute differences = 0

j=502
-x1- -x2-
152 152
133 133
114 114
95 95
76 76
57 57
38 38
19 19

Sum of absolute differences = 0

j=503
-x1- -x2-
144 144
126 126
108 108
90 90
72 72
54 54
36 36
18 18

Sum of absolute differences = 0

j=504
-x1- -x2-
136 136
119 119
102 102
85 85
68 68
51 51
34 34
17 17

Sum of absolute differences = 0

j=505
-x1- -x2-
128 128
112 112
96 96
80 80
64 64
48 48
32 32
16 16

Sum of absolute differences = 0

j=506
-x1- -x2-
120 120

```

105 105
90 90
75 75
60 60
45 45
30 30
15 15
Sum of absolute differences = 0

```

```

j=507
-x1- -x2-
112 112
98 98
84 84
70 70
56 56
42 42
28 28
14 14

```

Sum of absolute differences = 0

```

j=508
-x1- -x2-
104 104
91 91
78 78
65 65
52 52
39 39
26 26
13 13

```

Sum of absolute differences = 0

```

j=509
-x1- -x2-
96 96
84 84
72 72
60 60
48 48
36 36
24 24
12 12

```

Sum of absolute differences = 0

```

j=510
-x1- -x2-
88 88
77 77
66 66
55 55
44 44
33 33
22 22
11 11

```

Sum of absolute differences = 0

j=511

```

-x1- -x2-
80 80
70 70
60 60
50 50
40 40
30 30
20 20
10 10
Sum of absolute differences = 0

```

```

j=512
-x1- -x2-
72 72
63 63
54 54
45 45
36 36
27 27
18 18
9 9

```

Sum of absolute differences = 0
octave:63> who

*** currently compiled functions:

```

clock      columns  date      holes1
holes2     num2str  rem      rows

```

*** local user variables:

```

M  Np  err  j  p  x1  x2

```

```

octave:64> size(M)
ans =

```

```

1 1

```

```

octave:65> size(x1)
ans =

```

```

8 1

```

```

octave:66> size(err)
ans =

```

```

512 1

```

```

octave:67> max(abs(err))
ans = 0

```

```

octave:68> 'Simple hole finding function
works!'

```

```

ans = Simple hole finding function works!
octave:69> clock
ans =

```

```

octave:70> diary off

```

HOLES3.M

```
function [k,h] = holes3(y,p,M);

% h = holes3(y,p,M)
% CONFIDENTIAL AND PROPRIETARY
% Edwin A. Suominen
% Finds "holes" - skipped values of set  $\{0,1\}^N$  in result
% of  $x*y \bmod p$ .
% Uses equation discovered by EAS 9/16/00

% Number of values in set  $S:\{0,1\}^N$ 
%  $M = 2^N$ ;

k = p - (M+1);

% For vector inputs...
for i=1:length(y)
    for j=1:k,
        ## Input values between M+1 and p will of necessity
        ## be mapped to holes (values not produced by inputs
        ## from set  $\{1,2,\dots,M\}$  because  $xy \bmod p$  is a bijection
        ## (See HAC 1.8 Definition)
        ##  $h(j,i) = \text{rem}((M+j)*y, p)$ ;
        ## Equation above is simple but doesn't work when
        ##  $M < xy < p$  (which happens rarely, but it happens).

        h(j,i) = M+1 - rem(j*y(i)-k,p);
    endfor
endfor

if (nargout>=2)
    k = 1:k; k=k';
endif

endfunction
```

TEST3.M

TESTS EACH INPUT FOR ALL KEYS IN SPACE

```
## TEST3.M
## Block size is 10 bits. Input is taken from set Z:{1,2,...1024}
## Because of EAS-invented "pseudogroup" operation, output also
## falls in set Z.
## Keys are also taken from set Z - any set element is OK.

## This test proves the following:
## (1) Output set is same as input set Z.
## (2) Each input value has a unique output value, for a given
## key value.
## (3) The output value from "encrypt.m" can be converted back to
## the input value with "decrypt.m," given the key value.
## (4) For a given input value, each key value produces a unique
## output value.
## Written for Octave (GNU MATLAB alternative)

## No paging - want current screen output
page_screen_output=0;

## Set values defining set and underlying group order
N = 10; M = 2^N; # M = 1024
k = 7; p = M+k; # p = 1031 (prime)

## Create empty matrix of output values
A = zeros(M);

## Define vector with elements of set Z
v = linspace(1,M,M);

## Create string matrix of '-' neutral values for test condition codes
cc = ['-RESULTS- '; ' key: 1234']; # Header
## for each key value...
for i = 1:M
    ## insert key value before neutrals
    ccr = [num2str(i),': ----'];
    ## Leading zeros to make columns line up
    if i<10, ccr = ['0' ccr]; endif
    if i<100, ccr = ['0' ccr]; endif
    if i<1000, ccr = ['0' ccr]; endif
    cc(i+2,:) = ccr;
endfor

##### PART ONE OF TWO #####
disp(['Tests 1-3, for each key value in set 1,2,...',num2str(M)]);
disp('-----');
```

```
## For all possible key values in Z...
for i = 1:M
```

```
## Show progress
disp(['Encrypting and decrypting with key y=',num2str(i),'...']);
```

```
## Set key value for this iteration
y = v(i);
```

```
## Encrypt all possible input values in set Z with key
b = encrypt(v,y,N,k);
A(:,i) = b'; # Add this output vector to output matrix
```

```
## Test for conditions (1),(2) now
b = sort(b); # Sort ascending
```

```
disp(['Output set: min=',num2str(min(b)),', max=',num2str(max(b))]);
```

```
##### Test Condition (1) #####
```

```
if ( max(b)==M )
    disp('Output set is same as input set. ');
    cc(i+2,7) = '+';
else
    disp('PROBLEM: Output set larger or smaller than input set!');
    cc(i+2,7) = 'o';
endif
```

```
##### Test Condition (2) #####
```

```
## Each input value should have a unique output value, for a given
## key value.
```

```
b = diff(b); # Get differentials between sorted elements
```

```
if ( min(b)==1 & max(b)==1 )
```

```
    disp('All elements in output set are unique. ');
```

```
    cc(i+2,8) = '+';
```

```
else
```

```
    disp('PROBLEM: skipped or duplicated element(s) in output set!');
```

```
    cc(i+2,8) = 'o';
```

```
endif
```

```
##### Decrypt output values for this key #####
```

```
b = decrypt(A(:,i),y,N,k)';
```

```
##### Test Condition (3) #####
```

```

## Get differentials between plaintext-encrypted-decrypted (b) and
plaintext (v)
b = b - v; # Should be all zeros if test passes
if ( (max(abs(b))==0) )
    disp('All elements in input set encrypt and decrypt with key and
inverse.');
```

```

    cc(i+2,9) = '+';
else
    disp('PROBLEM: One or more elements do not match in
encryption/decryption!');
```

```

    cc(i+2,9) = 'o';
endif

disp('');
endifor

##### PART TWO OF TWO #####
disp(['Test 4, for each input value in set 1,2,...',num2str(M)]);
disp('-----');
```

```

##### Test Condition (4) #####
```

```

## For all possible input values in Z, working with full matrix of
outputs
for i = 1:M
```

```

    ## Show progress
    disp(['Analyzing outputs for input x=',num2str(i),' with all keys in
set...']);
```

```

    ## For a given input value, each key value should produce a unique
output value.
```

```

    b = diff(sort(A(i,:))); # Get differentials between sorted elements
    if ( min(b)==1 & max(b)==1 )
        disp('All elements in output set are unique.');
```

```

        cc(i+2,10) = '+';
    else
        disp('Skipped or duplicated element(s) in output set.');
```

```

        cc(i+2,10) = 'o';
    endif

    disp('');
endifor
```

```

## Display test results
disp(cc)
```

RESULTS OF TEST3.M

```
octave:14> date
ans = 
octave:15> clock
ans =
```

```
octave:16> type test3
test3 is the script file: /1093-2/test3.m
```

```
## TEST3.M
## Block size is 10 bits. Input is taken from set
Z:{1,2,...1024}
## Because of EAS-invented "pseudogroup" operation,
output also
## falls in set Z.
## Keys are also taken from set Z - any set element is
OK.
```

```
## This test proves the following:
## (1) Output set is same as input set Z.
## (2) Each input value has a unique output value, for a
given
## key value.
## (3) The output value from "encrypt.m" can be converted
back to
## the input value with "decrypt.m," given the key value.
## (4) For a given input value, each key value produces a
unique
## output value.
```

```
## Written for Octave (GNU MATLAB alternative)
```

```
## No paging - want current screen output
page_screen_output=0;
```

```
## Set values defining set and underlying group order
N = 10; M = 2^N; # M = 1024
k = 7; p = M+k; # p = 1031 (prime)
```

```
## Create empty matrix of output values
A = zeros(M);
```

```
## Define vector with elements of set Z
v = linspace(1,M,M);
```

```
## Create string matrix of '-' neutral values for test
condition codes
cc = ['-RESULTS- '; ' key: 1234']; # Header
## for each key value...
for i = 1:M
    ## insert key value before neutrals
    ccr = [num2str(i), '----'];
    ## Leading zeros to make columns line up
    if i<10, ccr = ['0' ccr]; endif
    if i<100, ccr = ['0' ccr]; endif
    if i<1000, ccr = ['0' ccr]; endif
    cc(i+2,:) = ccr;
endfor
```

```
##### PART ONE OF TWO #####
disp(['Tests 1-3, for each key value in set
1,2,...',num2str(M)]);
disp('-----');
----');
```

```
## For all possible key values in Z...
for i = 1:M
```

```
    ## Show progress
    disp(['Encrypting and decrypting with key
y=',num2str(i),'...']);
```

```
    ## Set key value for this iteration
    y = v(i);
```

```
    ## Encrypt all possible input values in set Z with key
    b = encrypt(v,y,N,k);
    A(:,i) = b'; # Add this output vector to output matrix
```

```
    ## Test for conditions (1),(2) now
    b = sort(b); # Sort ascending
```

```
    disp(['Output set: min=',num2str(min(b)),',
max=',num2str(max(b))]);
```

```
##### Test Condition (1) #####
```

```
if ( max(b)==M )
    disp('Output set is same as input set.');
```

```
    cc(i+2,7) = '+';
```

```
else
```

```
    disp('PROBLEM: Output set larger or smaller than
input set!');
```

```
    cc(i+2,7) = 'o';
```

```
endif
```

```
##### Test Condition (2) #####
```

```
## Each input value should have a unique output value,
for a given
```

```
## key value.
```

```
b = diff(b); # Get differentials between sorted
elements
```

```
if ( min(b)==1 & max(b)==1 )
```

```
    disp('All elements in output set are unique.');
```

```
    cc(i+2,8) = '+';
```

```
else
```

```
    disp('PROBLEM: skipped or duplicated element(s) in
output set!');
```

```
    cc(i+2,8) = 'o';
```

```
endif
```

```
##### Decrypt output values for this key #####
```

```
b = decrypt(A(:,i),y,N,k)';
```

```
##### Test Condition (3) #####
```

```
## Get differentials between plaintext-encrypted-
decrypted (b) and plaintext (v)
```

```
b = b - v; # Should be all zeros if test passes
```

```
if ( (max(abs(b))==0) )
```

```
    disp('All elements in input set encrypt and decrypt
with key and inverse.');
```

```
    cc(i+2,9) = '+';
```



```

else
    disp('PROBLEM: One or more elements do not match in
encryption/decryption!');
    cc(i+2,9) = 'o';
endif

disp('');

endfor

##### PART TWO OF TWO #####
disp(['Test 4, for each input value in set
1,2,...',num2str(M)]);
disp('-----');

#### Test Condition (4) ####

## For all possible input values in Z, working with full
matrix of outputs
for i = 1:M

    ## Show progress
    disp(['Analyzing outputs for input x=',num2str(i),'
with all keys in set...']);

    ## For a given input value, each key value should
    produce a unique output value.
    b = diff(sort(A(i,:))); # Get differentials between
sorted elements
    if (min(b)==1 & max(b)==1)
        disp('All elements in output set are unique. ');
        cc(i+2,10) = '+';
    else
        disp('Skipped or duplicated element(s) in output
set. ');
        cc(i+2,10) = 'o';
    endif

    disp('');

endfor

## Display test results
disp(cc)
octave:18> test3
Tests 1-3, for each key value in set 1,2,...1024
-----
Encrypting and decrypting with key y=1...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=2...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=3...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=4...
Output set: min=1, max=1024
Output set is same as input set.

```

All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=5...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=6...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=7...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=8...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=9...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=10...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=11...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=12...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=13...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

Encrypting and decrypting with key y=14...
Output set: min=1, max=1024
Output set is same as input set.
All elements in output set are unique.
All elements in input set encrypt and decrypt with key
and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in input set encrypt and decrypt with key and inverse.

All elements in output set are unique.


```

Analyzing outputs for input x=1013 with all keys in
set...
Skipped or duplicated element(s) in output set.

```

```
0001: ++++
0002: +++o
0003: +++o
0004: +++o
0005: +++o
0006: +++o
0007: +++o
0008: +++o
0009: +++o
0010: +++o
0011: +++o
0012: +++o
0013: +++o
0014: +++o
0015: +++o
0016: +++o
0017: +++o
0018: +++o
0019: +++o
0020: +++o
0021: +++o
0022: +++o
0023: +++o
0024: +++o
0025: +++o
0026: +++o
0027: +++o
```

0028: +++o
0029: +++o
0030: +++o
0031: +++o
0032: +++o
0033: +++o
0034: +++o
0035: +++o
0036: +++o
0037: +++o
0038: +++o
0039: +++o
0040: +++o
0041: +++o
0042: +++o
0043: +++o
0044: +++o
0045: +++o
0046: +++o
0047: +++o
0048: +++o
0049: +++o
0050: +++o
0051: +++o
0052: +++o
0053: +++o
0054: +++o
0055: +++o
0056: +++o
0057: +++o
0058: +++o
0059: +++o
0060: +++o
0061: +++o
0062: +++o
0063: +++o
0064: +++o
0065: +++o
0066: +++o
0067: +++o
0068: +++o
0069: +++o
0070: +++o
0071: +++o
0072: +++o
0073: +++o
0074: +++o
0075: +++o
0076: +++o
0077: +++o
0078: +++o
0079: +++o
0080: +++o
0081: +++o
0082: +++o
0083: +++o
0084: +++o
0085: +++o
0086: +++o
0087: +++o
0088: +++o
0089: +++o
0090: +++o
0091: +++o
0092: +++o
0093: +++o
0094: +++o
0095: +++o
0096: +++o
0097: +++o
0098: +++o
0099: +++o
0100: +++o

0101: +++o
0102: +++o
0103: +++o
0104: +++o
0105: +++o
0106: +++o
0107: +++o
0108: +++o
0109: +++o
0110: +++o
0111: +++o
0112: +++o
0113: +++o
0114: +++o
0115: +++o
0116: +++o
0117: +++o
0118: +++o
0119: +++o
0120: +++o
0121: +++o
0122: +++o
0123: +++o
0124: +++o
0125: +++o
0126: +++o
0127: +++o
0128: +++o
0129: +++o
0130: +++o
0131: +++o
0132: +++o
0133: +++o
0134: +++o
0135: +++o
0136: +++o
0137: +++o
0138: +++o
0139: +++o
0140: +++o
0141: +++o
0142: +++o
0143: +++o
0144: +++o
0145: +++o
0146: +++o
0147: +++o
0148: +++o
0149: +++o
0150: +++o
0151: +++o
0152: +++o
0153: +++o
0154: +++o
0155: +++o
0156: +++o
0157: +++o
0158: +++o
0159: +++o
0160: +++o
0161: +++o
0162: +++o
0163: +++o
0164: +++o
0165: +++o
0166: +++o
0167: +++o
0168: +++o
0169: +++o
0170: +++o
0171: +++o
0172: +++o
0173: +++o

0174: +++o
0175: +++o
0176: +++o
0177: +++o
0178: +++o
0179: +++o
0180: +++o
0181: +++o
0182: +++o
0183: +++o
0184: +++o
0185: +++o
0186: +++o
0187: +++o
0188: +++o
0189: +++o
0190: +++o
0191: +++o
0192: +++o
0193: +++o
0194: +++o
0195: +++o
0196: +++o
0197: +++o
0198: +++o
0199: +++o
0200: +++o
0201: +++o
0202: +++o
0203: +++o
0204: +++o
0205: +++o
0206: +++o
0207: +++o
0208: +++o
0209: +++o
0210: +++o
0211: +++o
0212: +++o
0213: +++o
0214: +++o
0215: +++o
0216: +++o
0217: +++o
0218: +++o
0219: +++o
0220: +++o
0221: +++o
0222: +++o
0223: +++o
0224: +++o
0225: +++o
0226: +++o
0227: +++o
0228: +++o
0229: +++o
0230: +++o
0231: +++o
0232: +++o
0233: +++o
0234: +++o
0235: +++o
0236: +++o
0237: +++o
0238: +++o
0239: +++o
0240: +++o
0241: +++o
0242: +++o
0243: +++o
0244: +++o
0245: +++o
0246: +++o

0247: +++o
0248: +++o
0249: +++o
0250: +++o
0251: +++o
0252: +++o
0253: +++o
0254: +++o
0255: +++o
0256: +++o
0257: +++o
0258: +++o
0259: +++o
0260: +++o
0261: +++o
0262: +++o
0263: +++o
0264: +++o
0265: +++o
0266: +++o
0267: +++o
0268: +++o
0269: +++o
0270: +++o
0271: +++o
0272: +++o
0273: +++o
0274: +++o
0275: +++o
0276: +++o
0277: +++o
0278: +++o
0279: +++o
0280: +++o
0281: +++o
0282: +++o
0283: +++o
0284: +++o
0285: +++o
0286: +++o
0287: +++o
0288: +++o
0289: +++o
0290: +++o
0291: +++o
0292: +++o
0293: +++o
0294: +++o
0295: +++o
0296: +++o
0297: +++o
0298: +++o
0299: +++o
0300: +++o
0301: +++o
0302: +++o
0303: +++o
0304: +++o
0305: +++o
0306: +++o
0307: +++o
0308: +++o
0309: +++o
0310: +++o
0311: +++o
0312: +++o
0313: +++o
0314: +++o
0315: +++o
0316: +++o
0317: +++o
0318: +++o
0319: +++o

0320: +++o
0321: +++o
0322: +++o
0323: +++o
0324: +++o
0325: +++o
0326: +++o
0327: +++o
0328: +++o
0329: +++o
0330: +++o
0331: +++o
0332: +++o
0333: +++o
0334: +++o
0335: +++o
0336: +++o
0337: +++o
0338: +++o
0339: +++o
0340: +++o
0341: +++o
0342: +++o
0343: +++o
0344: +++o
0345: +++o
0346: +++o
0347: +++o
0348: +++o
0349: +++o
0350: +++o
0351: +++o
0352: +++o
0353: +++o
0354: +++o
0355: +++o
0356: +++o
0357: +++o
0358: +++o
0359: +++o
0360: +++o
0361: +++o
0362: +++o
0363: +++o
0364: +++o
0365: +++o
0366: +++o
0367: +++o
0368: +++o
0369: +++o
0370: +++o
0371: +++o
0372: +++o
0373: +++o
0374: +++o
0375: +++o
0376: +++o
0377: +++o
0378: +++o
0379: +++o
0380: +++o
0381: +++o
0382: +++o
0383: +++o
0384: +++o
0385: +++o
0386: +++o
0387: +++o
0388: +++o
0389: +++o
0390: +++o
0391: +++o
0392: +++o

0393: +++o
0394: +++o
0395: +++o
0396: +++o
0397: +++o
0398: +++o
0399: +++o
0400: +++o
0401: +++o
0402: +++o
0403: +++o
0404: +++o
0405: +++o
0406: +++o
0407: +++o
0408: +++o
0409: +++o
0410: +++o
0411: +++o
0412: +++o
0413: +++o
0414: +++o
0415: +++o
0416: +++o
0417: +++o
0418: +++o
0419: +++o
0420: +++o
0421: +++o
0422: +++o
0423: +++o
0424: +++o
0425: +++o
0426: +++o
0427: +++o
0428: +++o
0429: +++o
0430: +++o
0431: +++o
0432: +++o
0433: +++o
0434: +++o
0435: +++o
0436: +++o
0437: +++o
0438: +++o
0439: +++o
0440: +++o
0441: +++o
0442: +++o
0443: +++o
0444: +++o
0445: +++o
0446: +++o
0447: +++o
0448: +++o
0449: +++o
0450: +++o
0451: +++o
0452: +++o
0453: +++o
0454: +++o
0455: +++o
0456: +++o
0457: +++o
0458: +++o
0459: +++o
0460: +++o
0461: +++o
0462: +++o
0463: +++o
0464: +++o
0465: +++o

0466: +++o
0467: +++o
0468: +++o
0469: +++o
0470: +++o
0471: +++o
0472: +++o
0473: +++o
0474: +++o
0475: +++o
0476: +++o
0477: +++o
0478: +++o
0479: +++o
0480: +++o
0481: +++o
0482: +++o
0483: +++o
0484: +++o
0485: +++o
0486: +++o
0487: +++o
0488: +++o
0489: +++o
0490: +++o
0491: +++o
0492: +++o
0493: +++o
0494: +++o
0495: +++o
0496: +++o
0497: +++o
0498: +++o
0499: +++o
0500: +++o
0501: +++o
0502: +++o
0503: +++o
0504: +++o
0505: +++o
0506: +++o
0507: +++o
0508: +++o
0509: +++o
0510: +++o
0511: +++o
0512: +++o
0513: +++o
0514: +++o
0515: +++o
0516: +++o
0517: +++o
0518: +++o
0519: +++o
0520: +++o
0521: +++o
0522: +++o
0523: +++o
0524: +++o
0525: +++o
0526: +++o
0527: +++o
0528: +++o
0529: +++o
0530: +++o
0531: +++o
0532: +++o
0533: +++o
0534: +++o
0535: +++o
0536: +++o
0537: +++o
0538: +++o

0539: +++o
0540: +++o
0541: +++o
0542: +++o
0543: +++o
0544: +++o
0545: +++o
0546: +++o
0547: +++o
0548: +++o
0549: +++o
0550: +++o
0551: +++o
0552: +++o
0553: +++o
0554: +++o
0555: +++o
0556: +++o
0557: +++o
0558: +++o
0559: +++o
0560: +++o
0561: +++o
0562: +++o
0563: +++o
0564: +++o
0565: +++o
0566: +++o
0567: +++o
0568: +++o
0569: +++o
0570: +++o
0571: +++o
0572: +++o
0573: +++o
0574: +++o
0575: +++o
0576: +++o
0577: +++o
0578: +++o
0579: +++o
0580: +++o
0581: +++o
0582: +++o
0583: +++o
0584: +++o
0585: +++o
0586: +++o
0587: +++o
0588: +++o
0589: +++o
0590: +++o
0591: +++o
0592: +++o
0593: +++o
0594: +++o
0595: +++o
0596: +++o
0597: +++o
0598: +++o
0599: +++o
0600: +++o
0601: +++o
0602: +++o
0603: +++o
0604: +++o
0605: +++o
0606: +++o
0607: +++o
0608: +++o
0609: +++o
0610: +++o
0611: +++o

0612: +++o
0613: +++o
0614: +++o
0615: +++o
0616: +++o
0617: +++o
0618: +++o
0619: +++o
0620: +++o
0621: +++o
0622: +++o
0623: +++o
0624: +++o
0625: +++o
0626: +++o
0627: +++o
0628: +++o
0629: +++o
0630: +++o
0631: +++o
0632: +++o
0633: +++o
0634: +++o
0635: +++o
0636: +++o
0637: +++o
0638: +++o
0639: +++o
0640: +++o
0641: +++o
0642: +++o
0643: +++o
0644: +++o
0645: +++o
0646: +++o
0647: +++o
0648: +++o
0649: +++o
0650: +++o
0651: +++o
0652: +++o
0653: +++o
0654: +++o
0655: +++o
0656: +++o
0657: +++o
0658: +++o
0659: +++o
0660: +++o
0661: +++o
0662: +++o
0663: +++o
0664: +++o
0665: +++o
0666: +++o
0667: +++o
0668: +++o
0669: +++o
0670: +++o
0671: +++o
0672: +++o
0673: +++o
0674: +++o
0675: +++o
0676: +++o
0677: +++o
0678: +++o
0679: +++o
0680: +++o
0681: +++o
0682: +++o
0683: +++o
0684: +++o

0685: +++o
0686: +++o
0687: +++o
0688: +++o
0689: +++o
0690: +++o
0691: +++o
0692: +++o
0693: +++o
0694: +++o
0695: +++o
0696: +++o
0697: +++o
0698: +++o
0699: +++o
0700: +++o
0701: +++o
0702: +++o
0703: +++o
0704: +++o
0705: +++o
0706: +++o
0707: +++o
0708: +++o
0709: +++o
0710: +++o
0711: +++o
0712: +++o
0713: +++o
0714: +++o
0715: +++o
0716: +++o
0717: +++o
0718: +++o
0719: +++o
0720: +++o
0721: +++o
0722: +++o
0723: +++o
0724: +++o
0725: +++o
0726: +++o
0727: +++o
0728: +++o
0729: +++o
0730: +++o
0731: +++o
0732: +++o
0733: +++o
0734: +++o
0735: +++o
0736: +++o
0737: +++o
0738: +++o
0739: +++o
0740: +++o
0741: +++o
0742: +++o
0743: +++o
0744: +++o
0745: +++o
0746: +++o
0747: +++o
0748: +++o
0749: +++o
0750: +++o
0751: +++o
0752: +++o
0753: +++o
0754: +++o
0755: +++o
0756: +++o
0757: +++o

0758: +++o
0759: +++o
0760: +++o
0761: +++o
0762: +++o
0763: +++o
0764: +++o
0765: +++o
0766: +++o
0767: +++o
0768: +++o
0769: +++o
0770: +++o
0771: +++o
0772: +++o
0773: +++o
0774: +++o
0775: +++o
0776: +++o
0777: +++o
0778: +++o
0779: +++o
0780: +++o
0781: +++o
0782: +++o
0783: +++o
0784: +++o
0785: +++o
0786: +++o
0787: +++o
0788: +++o
0789: +++o
0790: +++o
0791: +++o
0792: +++o
0793: +++o
0794: +++o
0795: +++o
0796: +++o
0797: +++o
0798: +++o
0799: +++o
0800: +++o
0801: +++o
0802: +++o
0803: +++o
0804: +++o
0805: +++o
0806: +++o
0807: +++o
0808: +++o
0809: +++o
0810: +++o
0811: +++o
0812: +++o
0813: +++o
0814: +++o
0815: +++o
0816: +++o
0817: +++o
0818: +++o
0819: +++o
0820: +++o
0821: +++o
0822: +++o
0823: +++o
0824: +++o
0825: +++o
0826: +++o
0827: +++o
0828: +++o
0829: +++o
0830: +++o

0831: +++o
0832: +++o
0833: +++o
0834: +++o
0835: +++o
0836: +++o
0837: +++o
0838: +++o
0839: +++o
0840: +++o
0841: +++o
0842: +++o
0843: +++o
0844: +++o
0845: +++o
0846: +++o
0847: +++o
0848: +++o
0849: +++o
0850: +++o
0851: +++o
0852: +++o
0853: +++o
0854: +++o
0855: +++o
0856: +++o
0857: +++o
0858: +++o
0859: +++o
0860: +++o
0861: +++o
0862: +++o
0863: +++o
0864: +++o
0865: +++o
0866: +++o
0867: +++o
0868: +++o
0869: +++o
0870: +++o
0871: +++o
0872: +++o
0873: +++o
0874: +++o
0875: +++o
0876: +++o
0877: +++o
0878: +++o
0879: +++o
0880: +++o
0881: +++o
0882: +++o
0883: +++o
0884: +++o
0885: +++o
0886: +++o
0887: +++o
0888: +++o
0889: +++o
0890: +++o
0891: +++o
0892: +++o
0893: +++o
0894: +++o
0895: +++o
0896: +++o
0897: +++o
0898: +++o
0899: +++o
0900: +++o
0901: +++o
0902: +++o
0903: +++o

0904: +++o
0905: +++o
0906: +++o
0907: +++o
0908: +++o
0909: +++o
0910: +++o
0911: +++o
0912: +++o
0913: +++o
0914: +++o
0915: +++o
0916: +++o
0917: +++o
0918: +++o
0919: +++o
0920: +++o
0921: +++o
0922: +++o
0923: +++o
0924: +++o
0925: +++o
0926: +++o
0927: +++o
0928: +++o
0929: +++o
0930: +++o
0931: +++o
0932: +++o
0933: +++o
0934: +++o
0935: +++o
0936: +++o
0937: +++o
0938: +++o
0939: +++o
0940: +++o
0941: +++o
0942: +++o
0943: +++o
0944: +++o
0945: +++o
0946: +++o
0947: +++o
0948: +++o
0949: +++o
0950: +++o
0951: +++o
0952: +++o
0953: +++o
0954: +++o
0955: +++o
0956: +++o
0957: +++o
0958: +++o
0959: +++o
0960: +++o
0961: +++o
0962: +++o
0963: +++o
0964: +++o
0965: +++o
0966: +++o
0967: +++o
0968: +++o
0969: +++o
0970: +++o
0971: +++o
0972: +++o
0973: +++o
0974: +++o
0975: +++o
0976: +++o

0977: +++o
0978: +++o
0979: +++o
0980: +++o
0981: +++o
0982: +++o
0983: +++o
0984: +++o
0985: +++o
0986: +++o
0987: +++o
0988: +++o
0989: +++o
0990: +++o
0991: +++o
0992: +++o
0993: +++o
0994: +++o
0995: +++o
0996: +++o
0997: +++o
0998: +++o
0999: +++o
1000: +++o
1001: +++o
1002: +++o
1003: +++o
1004: +++o
1005: +++o
1006: +++o
1007: +++o
1008: +++o
1009: +++o
1010: +++o
1011: +++o
1012: +++o
1013: +++o
1014: +++o
1015: +++o
1016: +++o
1017: +++o
1018: +++o
1019: +++o
1020: +++o
1021: +++o
1022: +++o
1023: +++o
1024: +++o
octave:19> diary off

RESULTS OF TEST3B.M

```
octave:4> date
ans = 
octave:5> clock
ans =
```

```
octave:6> type test3b
test3b is the script file: /1093-2/test3b.m
```

```
## TEST3B.M
## Block size is 10 bits. Input is taken from set Z:{1,2,...1024}
## Because of EAS-invented "pseudogroup" operation, output also
## falls in set Z.
## Keys are also taken from set Z - any set element is OK.

## This test analyzes outputs for a given input over all
## possible keys.
```

```
## Written for Octave (GNU MATLAB alternative)
```

```
## No paging - want current screen output
page_screen_output=0;
```

```
## Set values defining set and underlying group order
N = 10; M = 2^N; # M = 1024
k = 7; p = M+k; # p = 1031 (prime)
```

```
## Define vector with elements of set Z
v = linspace(1,M,M);
```

```
## Define vector of skip/repeat counts
cc = zeros(1,M);
```

```
disp(['Test for each input value in set 1,2,...',num2str(M)]);
disp('-----');
```

```
## For all possible input values in Z...
for i = 1:M
```

```
    ## Show progress
    disp(['Encrypting with input value y=',num2str(i),'...']);
```

```
    ## Set input value for this iteration
    x = v(i);
```

```
    ## Encrypt input value with all keys in set Z
    for j = 1:M
        b(j) = encrypt(x,v(j),N,k);
    endfor
```

```
disp(['Output set: min=',num2str(min(b)),', max=',num2str(max(b))]);
disp('');
```

```
## Identify any skipped or repeated set elements
## with vector of index numbers
b1 = sort(b); # Sort ascending
b2 = [diff(b1)']; # Should be all 1's...
b2 = b2~=1; # ...so 1's indicate skips/repeats
```

```
Nsr = sum(b2); # Count of skips/repeats
```

```
b3 = b2 .* v(1:M-1); # map index numbers to skips/repeats
b3 = sort(b3); # Sort ascending
b4 = b3(M-Nsr:M-1); # Select only skips/repeats
```

```
if (Nsr > 0)
```

```
disp(['There are ',num2str(Nsr),' skips & repeats, at:']);
disp(b4);
disp('-----');
```

```
c = zeros(6,Nsr); # Start with empty ("0") matrix
for j = 1:Nsr
    k1 = max([1 b4(j)-2]);
    k2 = min([b4(j)+3 M]);
    c(1:k2-k1+1,j) = b1(k1:k2);
endfor
```

```
disp(c)
```

```
endif
```

```
cc(i) = Nsr; # Add this count to vector
disp(['Maximum skips & repeats for a given input (so far):',num2str(max(cc))]);
disp('');
```

```
endfor
```

```
...
```

```
octave:9> test3b
```

```
Test for each input value in set 1,2,...1024
```

```
-----
Encrypting with input value y=1...
```

```
Output set: min=1, max=1024
```

```
Maximum skips & repeats for a given input (so far):0
```

```
Encrypting with input value y=2...
```

```
Output set: min=1, max=1024
```

```
There are 6 skips & repeats, at:
```

```
10  518  520  1021  1022  1023
```

```
-----
8   515  517  1016  1017  1018
```

```
9   516  517  1017  1018  1020
```

```
10  517  518  1018  1020  1022
```

```
10  517  518  1020  1022  1024
```

11	518	519	1022	1024	0
12	518	520	1024	0	0

Maximum skips & repeats for a given input (so far):6

Encrypting with input value y=3...
Output set: min=1, max=1024

There are 8 skips & repeats, at:

10	346	690	692	1016	1018	1020	1022

8	343	686	688	1010	1012	1015	1018
9	344	687	688	1011	1014	1017	1020
10	345	688	689	1012	1015	1018	1021
10	345	688	689	1014	1017	1020	1023
11	346	689	690	1015	1018	1021	1024
12	347	689	691	1017	1020	1023	0

Maximum skips & repeats for a given input (so far):8

Encrypting with input value y=4...
Output set: min=1, max=1024

There are 10 skips & repeats, at:

3	260	519	523	778	1011	1014	1017	1020	1023

1	257	515	518	772	1004	1008	1012	1016	1020
2	258	516	519	773	1005	1009	1013	1017	1021
3	259	517	520	774	1006	1010	1014	1018	1022
3	259	517	520	774	1008	1012	1016	1020	1024
4	260	518	521	775	1009	1013	1017	1021	0
5	261	519	522	776	1010	1014	1018	1022	0

Maximum skips & repeats for a given input (so far):10

Encrypting with input value y=5...
Output set: min=1, max=1024

There are 10 skips & repeats, at:

6	212	418	624	830	1005	1009	1013	1017	1021

4	209	414	619	824	998	1003	1008	1013	1018
5	210	415	620	825	999	1004	1009	1014	1019
6	211	416	621	826	1000	1005	1010	1015	1020
6	211	416	621	826	1002	1007	1012	1017	1022
7	212	417	622	827	1003	1008	1013	1018	1023
8	213	418	623	828	1004	1009	1014	1019	1024

Maximum skips & repeats for a given input (so far):10

Encrypting with input value y=6...
Output set: min=1, max=1024

There are 10 skips & repeats, at:

176	346	518	691	864	999	1004	1009	1014	1019

174	343	514	686	858	992	998	1004	1010	1016
175	344	515	687	859	993	999	1005	1011	1017
176	345	516	688	860	994	1000	1006	1012	1018
176	345	516	688	860	996	1002	1008	1014	1020
177	346	517	689	861	997	1003	1009	1015	1021
178	347	518	690	862	998	1004	1010	1016	1022

Maximum skips & repeats for a given input (so far):10

Encrypting with input value y=7...

Output set: min=1, max=1023

There are 11 skips & repeats, at:

149	297	445	593	741	889	994	1000	1006	1012	1018
147	294	441	588	735	882	986	993	1000	1007	1014
148	295	442	589	736	883	987	994	1001	1008	1015
149	296	443	590	737	884	988	995	1002	1009	1016
149	296	443	590	737	884	990	997	1004	1011	1018
150	297	444	591	738	885	991	998	1005	1012	1019
151	298	445	592	739	886	992	999	1006	1013	1020

Maximum skips & repeats for a given input (so far):11

Encrypting with input value y=8...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

3	4	261	520	521	779	988	995	1002	1009	1016	1023
1	2	257	515	516	772	980	988	996	1004	1012	1020
2	3	258	516	517	773	981	989	997	1005	1013	1021
3	3	259	517	517	774	982	990	998	1006	1014	1022
3	3	259	517	517	774	984	992	1000	1008	1016	1024
3	4	260	517	518	775	985	993	1001	1009	1017	0
4	5	261	518	519	776	986	994	1002	1010	1018	0

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=9...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

2	346	347	464	692	922	982	990	998	1006	1014	1022
1	343	344	459	686	915	974	983	992	1001	1010	1019
2	344	345	460	687	916	975	984	993	1002	1011	1020
2	345	345	461	688	917	976	985	994	1003	1012	1021
3	345	345	461	688	917	978	987	996	1005	1014	1023
4	345	346	462	689	918	979	988	997	1006	1015	1024
0	346	347	463	690	919	980	989	998	1007	1016	0

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=10...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

2	4	416	520	829	933	976	985	994	1003	1012	1021
1	2	412	515	823	926	968	978	988	998	1008	1018
2	2	413	516	824	927	969	979	989	999	1009	1019
2	3	414	517	825	928	970	980	990	1000	1010	1020
3	3	414	517	825	928	972	982	992	1002	1012	1022
3	4	415	518	826	929	973	983	993	1003	1013	1023
0	5	416	519	827	930	974	984	994	1004	1014	1024

Maximum skips & repeats for a given input (so far):12

17	35	53	71	89	107	114	342	458	687	687	859
19	37	55	73	91	109	114	342	458	687	687	859
20	38	56	74	92	110	115	343	459	687	688	860
21	39	57	75	93	111	116	344	460	688	689	861

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1014...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

16	32	48	64	80	96	236	479	661	845	907	969

14	31	48	65	82	99	240	482	663	846	907	968
15	32	49	66	83	100	241	483	664	847	908	969
16	33	50	67	84	101	242	484	665	848	909	970
18	35	52	69	86	103	242	484	665	848	909	970
19	36	53	70	87	104	243	485	666	849	910	971
20	37	54	71	88	105	244	486	667	850	911	972

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1015...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

15	30	45	60	75	90	121	188	511	641	771	901

13	29	45	61	77	93	125	191	513	642	771	900
14	30	46	62	78	94	126	192	514	643	772	901
15	31	47	63	79	95	127	193	515	644	773	902
17	33	49	65	81	97	127	193	515	644	773	902
18	34	50	66	82	98	128	194	516	645	774	903
19	35	51	67	83	99	129	195	517	646	775	904

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1016...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

14	28	42	56	70	84	199	338	614	752	891	961

12	27	42	57	72	87	203	341	616	753	891	960
13	28	43	58	73	88	204	342	617	754	892	961
14	29	44	59	74	89	205	343	618	755	893	962
16	31	46	61	76	91	205	343	618	755	893	962
17	32	47	62	77	92	206	344	619	756	894	963
18	33	48	63	78	93	207	345	620	757	895	964

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1017...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

13	26	39	52	65	78	140	436	437	658	734	956

11	25	39	53	67	81	144	439	440	659	734	955
12	26	40	54	68	82	145	440	441	660	735	956
13	27	41	55	69	83	146	441	441	661	736	957
15	29	43	57	71	85	146	441	441	661	736	957

16 30 44 58 72 86 147 441 442 662 737 958
 17 31 45 59 73 87 148 442 443 663 738 959
 Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1018...
 Output set: min=1, max=1024

There are 12 skips & repeats, at:
 12 24 36 48 60 72 73 231 233 631 632 871

 10 23 36 49 62 75 76 234 236 632 633 870
 11 24 37 50 63 76 77 235 236 633 634 871
 12 25 38 51 64 77 77 236 237 634 634 872
 14 27 40 53 66 77 79 236 237 634 634 872
 15 28 41 54 67 79 80 237 238 634 635 873
 16 29 42 55 68 80 81 237 239 635 636 874
 Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1019...
 Output set: min=1, max=1024

There are 12 skips & repeats, at:
 11 22 33 44 55 66 336 508 682 769 857 944

 9 21 33 45 57 69 340 511 684 770 857 943
 10 22 34 46 58 70 341 512 685 771 858 944
 11 23 35 47 59 71 342 513 686 772 859 945
 13 25 37 49 61 73 342 513 686 772 859 945
 14 26 38 50 62 74 343 514 687 773 860 946
 15 27 39 51 63 75 344 515 688 774 861 947
 Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1020...
 Output set: min=1, max=1024

There are 12 skips & repeats, at:
 10 20 30 40 50 60 85 182 275 652 747 936

 8 19 30 41 52 63 89 185 277 653 747 935
 9 20 31 42 53 64 90 186 278 654 748 936
 10 21 32 43 54 65 91 187 279 655 749 937
 12 23 34 45 56 67 91 187 279 655 749 937
 13 24 35 46 57 68 92 188 280 656 750 938
 14 25 36 47 58 69 93 189 281 657 751 939
 Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1021...
 Output set: min=1, max=1024

There are 12 skips & repeats, at:
 9 18 27 36 45 54 199 407 510 615 616 823

 7 17 27 37 47 57 203 410 512 616 617 822
 8 18 28 38 48 58 204 411 513 617 618 823
 9 19 29 39 49 59 205 412 514 618 618 824
 11 21 31 41 51 61 205 412 514 618 618 824
 12 22 32 42 52 62 206 413 515 618 619 825
 13 23 33 43 53 63 207 414 516 619 620 826

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1022...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

8	16	24	32	40	48	222	336	567	684	799	915

6	15	24	33	42	51	226	339	569	685	799	914
7	16	25	34	43	52	227	340	570	686	800	915
8	17	26	35	44	53	228	341	571	687	801	916
10	19	28	37	46	55	228	341	571	687	801	916
11	20	29	38	47	56	229	342	572	688	802	917
12	21	30	39	48	57	230	343	573	689	803	918

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1023...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

7	14	21	28	35	42	248	249	511	641	771	901

5	13	21	29	37	45	252	253	513	642	771	900
6	14	22	30	38	46	253	254	514	643	772	901
7	15	23	31	39	47	254	254	515	644	773	902
9	17	25	33	41	49	254	254	515	644	773	902
10	18	26	34	42	50	254	255	516	645	774	903
11	19	27	35	43	51	255	256	517	646	775	904

Maximum skips & repeats for a given input (so far):12

Encrypting with input value y=1024...

Output set: min=1, max=1024

There are 12 skips & repeats, at:

6	12	18	24	30	36	137	286	435	584	733	882

4	11	18	25	32	39	141	289	437	585	733	881
5	12	19	26	33	40	142	290	438	586	734	882
6	13	20	27	34	41	143	291	439	587	735	883
8	15	22	29	36	43	143	291	439	587	735	883
9	16	23	30	37	44	144	292	440	588	736	884
10	17	24	31	38	45	145	293	441	589	737	885

Maximum skips & repeats for a given input (so far):12

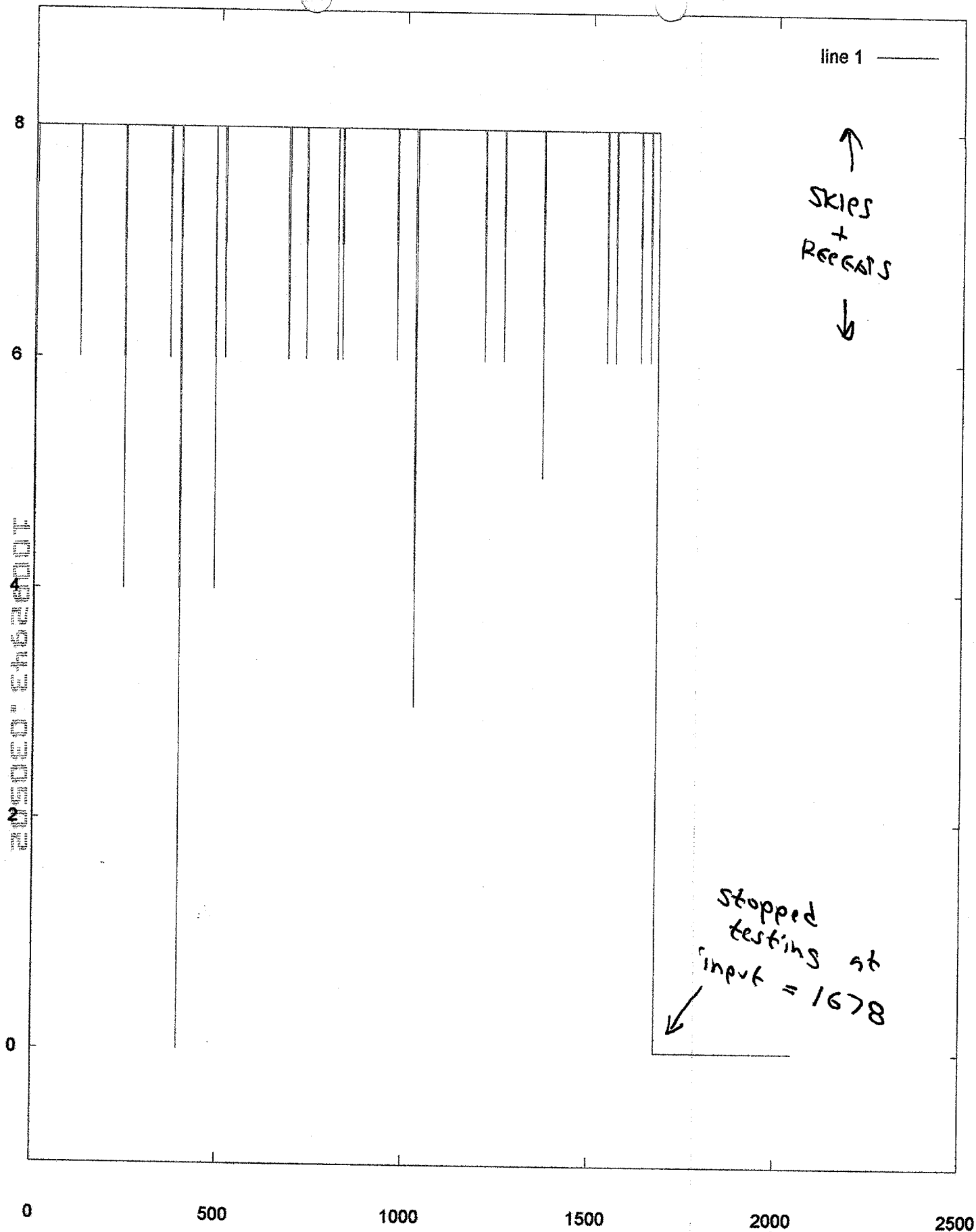
octave:10> date

ans =

octave:11> clock

ans =

octave:12> diary off



TGST4B.M
WITH M = 2048
p = 2053

Digit Content
 Alternating consonants (C), Vowels (V)
 C,13: {b,d,g,h,k,l,m,n,p,r,s,t,z} V,5: {a,e,i,o,u}

by Edwin A. Suominen

Digit Content	Min. Digits	Entropy	Example Passphrase
Alternating consonants (C), Vowels (V)	C: 11	About 64 bits, for	nihudezo dogiz pozubume
C,13: {b,d,g,h,k,l,m,n,p,r,s,t,z} V,5: {a,e,i,o,u}	V: 10	minimum digits	

The vowels and phonetically distinct consonanants below are pseudorandomly distributed, created using alternating pseudorandom lookups to a list of consonants b,d,g,h,k,l,m,n,p,r,s,t,z and vowels a,e,i,o,u. Alternating randomly selected consonants and randomly selected vowels from the array below form passphrases that have a faintly Oriental or African sound to them, and are more memorable than random alphanumerices. Consonants c,f,j,q,v,w,x,y are omitted because the passphrases tend to have a more distinct sound and are easier to pronounce without them.

You should split the consonant/vowel pairs into groups to make the passphrase pronounceable and thus more memorable. The suggested way of grouping the minimum 11 consonants and 10 vowels is as follows: CVCVCVCV CVCVC CVCVCVCV. Note that the middle group begins and ends with a consonant. The resulting passphrase has a distinct sound that makes you wonder if the “words” show up in some foreign language even though they’re just groups of randomly chosen letters.

Unbend a paper clip slightly, repeatedly toss the clip onto a printout of this page without aiming it anywhere in particular, and select the consonant/vowel pair to which the unbent end comes closest to get the next digit in your consonant-vowel sequence. Don't use both digits from a pair – each digit in your passphrase needs its own toss. With good random tosses, you can expect the clip to bounce outside the array of digits below about half the time. Just toss again. Don't aim at any particular region.

ne ze nu hi ni ba hu pi pu lu me ki lu ge le mu nu ko se ze ta ba ga ro ta be ko te
lo ru hu gi ra ga ro do ro zu tu si mu pu lo bi ze tu ha ti ru to ni ze bu be si ma
ro le hi ni bo ma do hu ma lu bu zi ru sa ta zo po go ta hi ri bi te la me go ho si
go ka hi bo li ro mi ta mo ki ku hu ri le da ra za mi le mi da ra zu ki ke bi mi do
mo ta tu ta ra si la go ki su ki mi re ba le so to za ba te hi ri da go za ka sa za
mo na la me ku bu mu me no ke ri be ho tu mu ki no re ho pe mi hu ho pi ge ro to se
go ka ze ri du sa de bi si de mi pi se po le mi bu re pe du so lu la ri mo mo bo go
to ge ne go lo ru sa da ma ga ni pe si se bu di pi da pu ta la ku hu ko go za si de
tu me po ro to pe ku hu ra ha bu zo ti na ba ma se te pi do to ke so do re ru ze ru
to lo ne ki ta ha go go li ra po de na du ba su po ka lu bu hi be za la ke bo ne mi
he ni te ma ni pu ho ru ne da nu de ge ge so ge lu re ho bi po li ma ri su pa te du
zi ta ma lu ti bu ha ba hu mu su za ko to ga te tu ru la ki ru ze po ni pi ho sa pa
ba bu pi za du go ga su mu re su ni re hi bi ko po la zu ri ku ka zi ba pa tu di da
ma mo di hu po ro ho ku se pa bo la ga ne me pi pu gu he li ha gi bo he hu so ru pa
se be pe ga se ni lo bo se ka be gi de zi di me mu lu pe ku tu go pe hi di bi de ne
mo mi ki ni ka ha nu ka za do he ni da du ri se du lo se tu pe to ti hu na ru se ta
na ho bu tu ze ge ru ru ba ga li ro so mu du he da go mu da ne tu mu pi ne bu ka la
li le he me gi ru ge hi lo po ho pe no mu pe ta lo re na di gi ko ze re pi ga ba ki
ke su zo ra no di lu du be za na po do su ra do si he me bi ga ra ha do za ru ku ri
zi bu do lo ga ki ha zi ka ru re ke ti lo zu zu gu bu ma bo hi he si du ne hu da ka
te ne na zi hu si da no mo te bu ko tu ro mu ne ma su li ba ga te se mo bi re si tu
ta pe bo bu ma tu ge be ri no pa do ri za ra ho so to da ze ze lo he mi ha la de me
pu ha no te de la zi hu gi ro te pa mo bu mu pe da sa pe na pi ti ru ro pi go li ku
gi gi pu za pe go ti lo pu du su mo ta ki ha ge ka ke pa tu po hi be lu sa bo li so
ko ra ri zi po ra la ze pe ru zu ra mu zu du ka lo ku zo po mo no he ze su ho le gi
te go pu ka ga la ki lu pa ti na ga gi ba ka pi ka su gi mo no si te li gi se zi ze
ze za lu ge ro zi ra do bo ma zo mi ke ze la bu ru be ki te ga ni go gi ga no ha
si bu be mi ta pa de ke ta lu ru ma ra si se te me pe pu gi mu me li bi bi hi zu me
de he ku zu nu gi pi zi ga ka ze lo re ru ha zu ta ku pi po me la da mu li de he be
pi le mu ko ki nu ke ga hu la la ri po su ri na ru no ta ro ho zi so ru ri go su hu
gi ra ru pu mu gi le ga ti ne go le ba da gi si si ni ra zu ma ha bu le se ze ro go
pi no bu ne nu ma si re ka gu zo si bo sa be zo si ko po li gi ra pe ne gu ri la ki
go ka ze ne ro mo bi nu he no hu ne ha lo ne le hu hi so du de li ta ze li hu de do
ga ha bi he su ku pe de he hu ta so zo he bu ti hi gi ta ti ze pe gu ni ka ra su pi
lu ge pe ka pu na da mu ti du mo pi de pu pa ri si bi sa mi li ko mi be ge ku su po
me mi tu ze ge ki gu hu hi lo tu da mu to ru gi ru ra mo na ma ba si se ri ro ro ka
po ge do sa pu ma so na ba la li ka ne ba di te zo so ru mi se ni gu na ta di zi ri
ga ga ri no ho sa ne mo le hi he re ta ne go ki de pi de gi mu du tu bo to ki po di
ma ge bo se ne hi ni ma go li le ra ri mu si lo gi zo ku bo ro ze po to do go re tu
gi ka la ki za ti me te zi ho mi ta no ne lo za ma mo te si sa ba da ku ko be ki go
zi zu tu pe ma ba zu lo ra ge hu pa te ra ra ko zu ni se he to ma se gu gu ba bi te
pe zo ne no gu ne bi ga ra na ra na la la do nu du za ro lili mu ne ru ni he te ma

Description	Digit Space #1	No. of #1 Digits	Entropy #1 (bits)	Digit Space #2	No. of #2 Digits	Entropy #2 (bits)	Digit Space #3	No. of #3 Digits	Entropy #3 (bits)	Total Entropy
6 word DICEWARE (http://dice.ware.com)	7776	6	77.54888	1	0	0	0	1	0	77.5
14 alphanumeric digits except lowercase "L" (can group 5-4-5 or 4-6-4)	35	14	71.80996	1	0	0	0	1	0	71.8
13 alphanumeric digits except lowercase "L" (can group 4-5-4)	35	13	66.68068	1	0	0	0	1	0	66.7
Pairs of phonetically distinct consonants "b,d,g,h,k,l,m,n,p,r,s,t,z") followed by vowel, arranged as follows: cvcvvcvc cvvc cvcvvcvc	13	11	40.70484	5	10	23.21928	1	0	0	63.9
12 alphanumeric digits except lowercase "L" separated randomly into three groups	35	12	61.5514	11	1	3.459432	1	0	0	65.0
5 word DICEWARE	7776	5	64.62406	1	0	0	0	1	0	64.6
12 alphanumeric digits except lowercase "L" (can be grouped 4-4-4)	35	12	61.5514	1	0	0	0	1	0	61.6
4 words from 70K dictionary	70000	4	64.38027	1	0	0	0	1	0	64.4
Three dates, Month, Day, Year	12	3	10.75489	30	3	14.72067	500	3	26.89735	52.4
Random first name (10K) plus M.I. Plus last name (50K)	10000	1	13.28771	50000	1	15.60964	26	1	4.70044	33.6
Random street address	20000	1	14.28771	8	1	3	10000	1	13.28771	30.6
++ No.				NSEW		30.6	20	1	4.321928	34.9
(e.s.,	1	0	0	0	0	0	0	1	0	0.0
1	1	0	0	0	0	0	0	1	0	0.0
Passwords for smartcards										
Pairs of phonetically distinct consonants "b,d,g,h,k,l,m,n,p,r,s,t,z") followed by vowel, arranged as follows: cvcvvcvc cvvc cvcvvcvc	13	4	14.80176	5	4	9.287712	1	0	0	24.1
5 alphanumeric digits except lowercase "L"	35	5	25.64642	1	0	0	1	0	0	25.6
(e.s.,	1	0	0	0	0	0	0	1	0	0.0
1	1	0	0	0	0	0	0	1	0	0.0
Medium Security for use with secure delay										
Pairs of phonetically distinct consonants "b,d,g,h,k,l,m,n,p,r,s,t,z") followed by vowel, arranged as follows: cvcvvcvc cvvc cvcvvcvc	13	8	29.60352	5	8	18.57542	1	0	0	48.2
Pairs of phonetically distinct consonants "b,d,g,h,k,l,m,n,p,r,s,t,z") followed by vowel, arranged as follows: cvcvvcvc cvvc cvcvvcvc	10	8	26.57542	5	8	18.57542	1	0	0	45.2
Groups of numbers 1000-9192 = {0-8192}+1000	8192	4	52	1	0	0	1	0	0	52
1	1	0	0	0	0	0	0	1	0	0.0
1	1	0	0	0	0	0	0	1	0	0.0

SECURE PASSPHRASE

Z2-Z6 illustrate screen shots of a secure passphrase entry system according to various aspects of the invention, illustrating an exemplary user interface at different points during the input of a passphrase without the use of keystrokes. Thus, the security hazard of keystroke loggers can be avoided. In addition, the mouse-based input method may be preferred by users over the use of a keyboard, for example when they are entering their passphrase to browse encrypted e-mails or files. In an experiment the applicant carried out, "entering" the passphrase by the mouse input method (simulated by tapping a pen onto a printout similar to Z2-Z6) did not take him much longer than typing in the passphrase.

Advantageously, the passphrase is represented in the illustrated embodiment (as it is entered) both as circled letters and has a pair of stair-stepped line segments having characteristic shapes. Viewing the passphrase and its associated characteristic shapes of the line segments helps the user to remember the passphrase. Human brains are good at remembering pronounceable words (even when they are nonsense words) and are also good at remembering characteristic shapes. The combination of both characteristics of a unique passphrase can be expected to improve the user's ability to remember it when the time comes to input the passphrase.

A delay system according to another aspect of the invention, illustrated in the block diagrams of Z7 and Z8, makes a secure delay according to various aspects of the invention less unobtrusive to the user. It does so by beginning the delay process when the passphrase has been partially entered. Advantageously, such a system performs the delay computations substantially in parallel with the unavoidable delay of the user's input of the passphrase. Even when typing quickly, it took the applicant at least about three seconds to enter the passphrase during his experiment. This is a substantial period of delay that, when made computationally unavoidable, makes cracking the 2^{48} possible combinations of the randomly chosen passphrase nearly impossible with the computing horsepower available around the date of filing of the present application. (See Z9 and Z10 for a detailed computational analysis.) The screen shots of Z2-Z6 show the "private key delayed unlocking" beginning with the first consonant-vowel pair entered by the user. The delayed unlocking (the inventive "computationally unavoidable" delay) continues substantially in parallel with the user's input of additional consonant-vowel pairs. Note Z6, in which the passphrase is confirmed and the private key has been completely unlocked.

###

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

N	I	H	—	C	V	C	V	C	V	C	V
---	---	---	---	---	---	---	---	---	---	---	---

Click on the first letter of your passphrase and move your mouse pointer up and down over the next column of letters to move the arrow to the second letter of the passphrase. Then click and repeat until you've entered the entire passphrase. If you prefer, you can drag the arrow line through all letters of the passphrase without releasing your mouse button.

If you are certain your computer is secure and your keystrokes are not being logged somehow, you can also type your passphrase in the text box above.

Private Key Delayed Unlock

[illegible]

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

A E I O U

B D G H K L M N P R S T Z

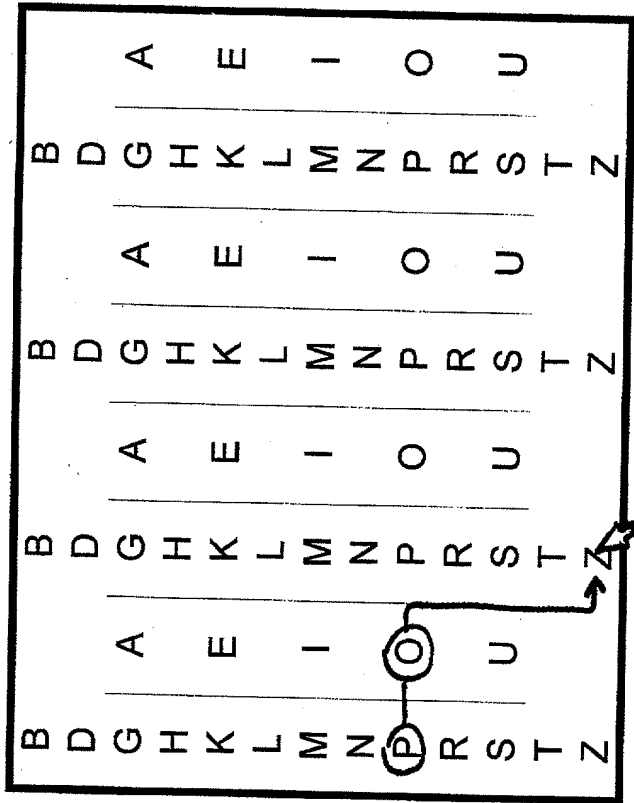
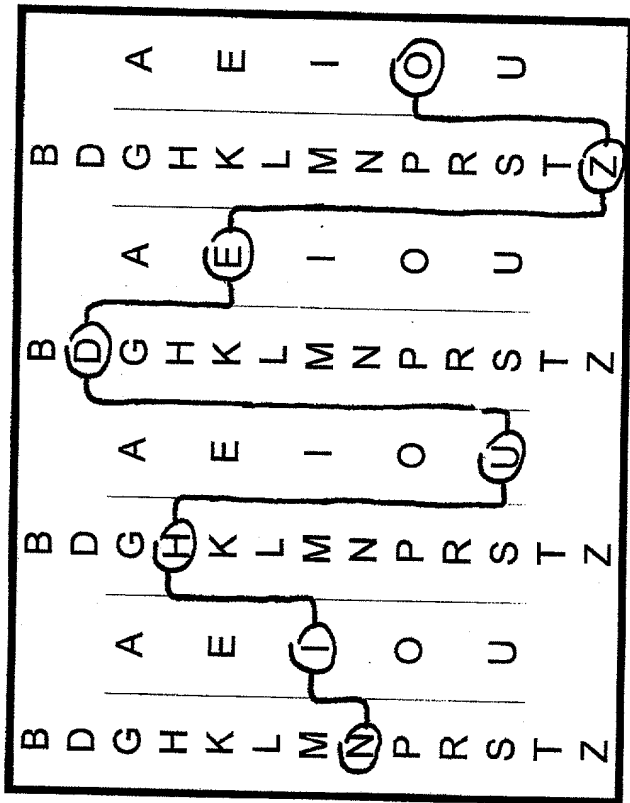
N I H U D _ C V C V C V C V C V

Click on the first letter of your passphrase and move your mouse pointer up and down over the next column of letters to move the arrow to the second letter of the passphrase. Then click and repeat until you've entered the entire passphrase. If you prefer, you can drag the arrow line thorough all letters of the passphrase without releasing your mouse button.

If you are certain your computer is secure and your keystrokes are not being logged somehow, you can also type your passphrase in the text box above.

Private Key Delayed Unlock

[illegible]



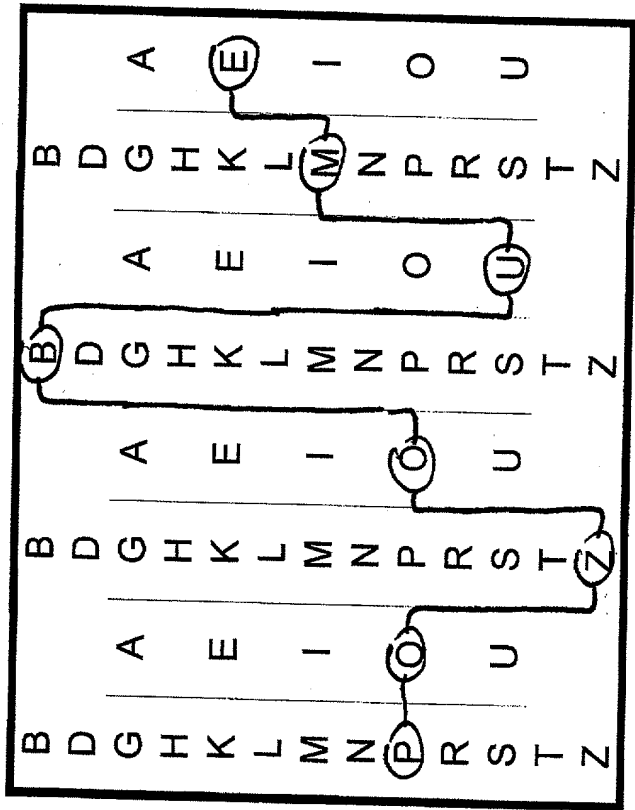
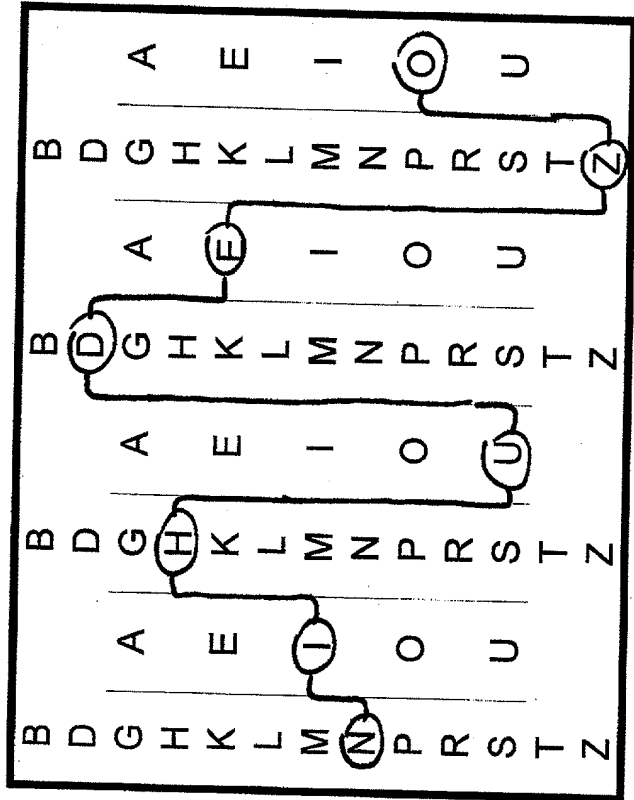
N I H U D E Z O P O _ V C V C V

Click on the first letter of your passphrase and move your mouse pointer up and down over the next column of letters to move the arrow to the second letter of the passphrase. Then click and repeat until you've entered the entire passphrase. If you prefer, you can drag the arrow line through all letters of the passphrase without releasing your mouse button.

If you are certain your computer is secure and your keystrokes are not being logged somehow, you can also type your passphrase in the text box above.

Private Key Delayed Unlock

--	--	--	--	--	--	--	--	--	--



N I H U D E Z O P O Z O B U M E

Click on the first letter of your passphrase and move your mouse pointer up and down over the next column of letters to move the arrow to the second letter of the passphrase. Then click and repeat until you've entered the entire passphrase. If you prefer, you can drag the arrow line thorough all letters of the passphrase without releasing your mouse button.

If you are certain your computer is secure and your keystrokes are not being logged somehow, you can also type your passphrase in the text box above.

Private Key Delayed Unlock

B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								

B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								
B	D	G	H	K	L	M	N	P	R	S	T	Z
A	E	I	O	U								

Passphrase confirmed. Your signature has been applied.

Click on the first letter of your passphrase and move your mouse pointer up and down over the next column of letters to move the arrow to the second letter of the passphrase. Then click and repeat until you've entered the entire passphrase. If you prefer, you can drag the arrow line through all letters of the passphrase without releasing your mouse button.

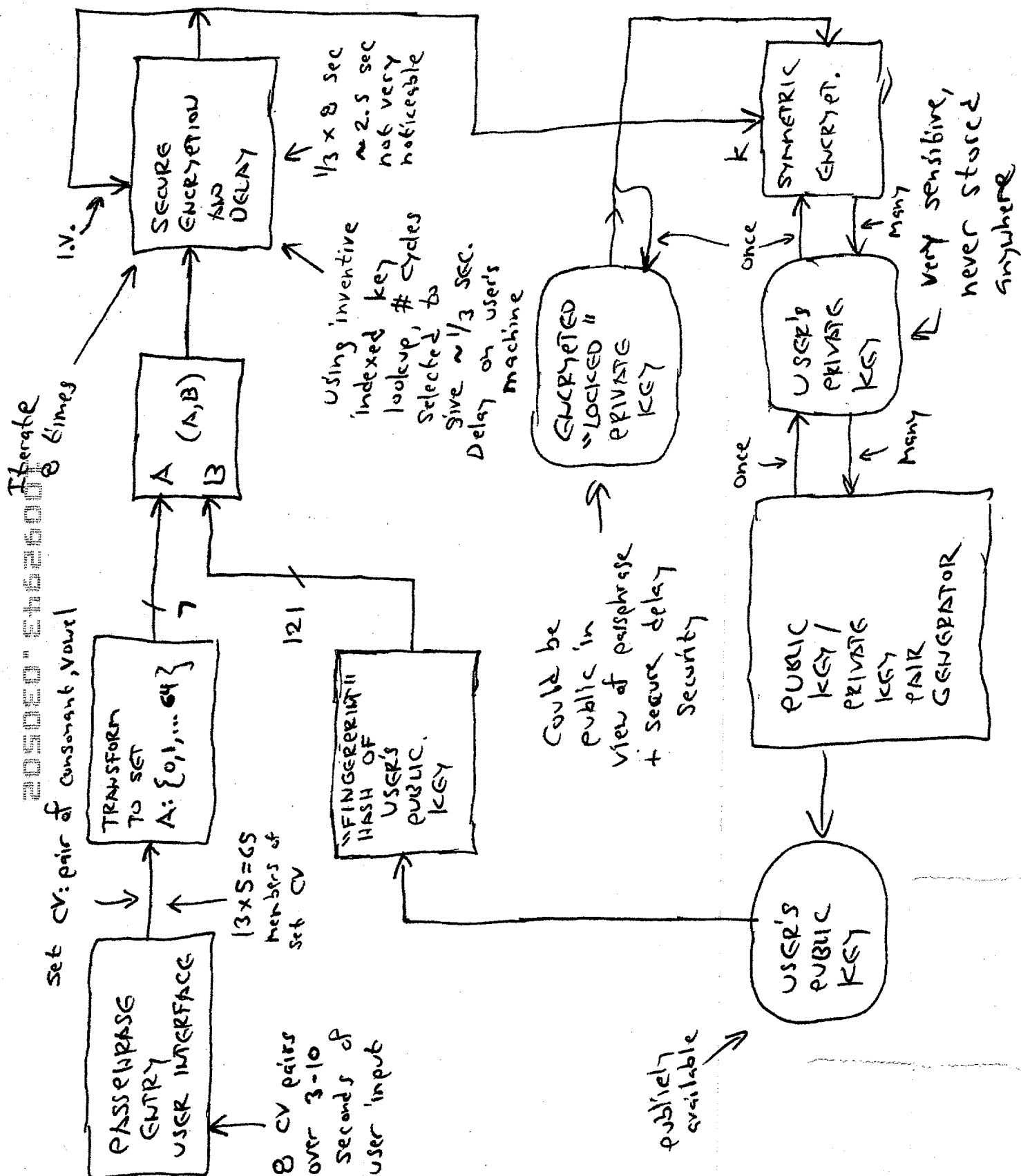
If you are certain your computer is secure and your keystrokes are not being logged somehow, you can also type your passphrase in the text box above.

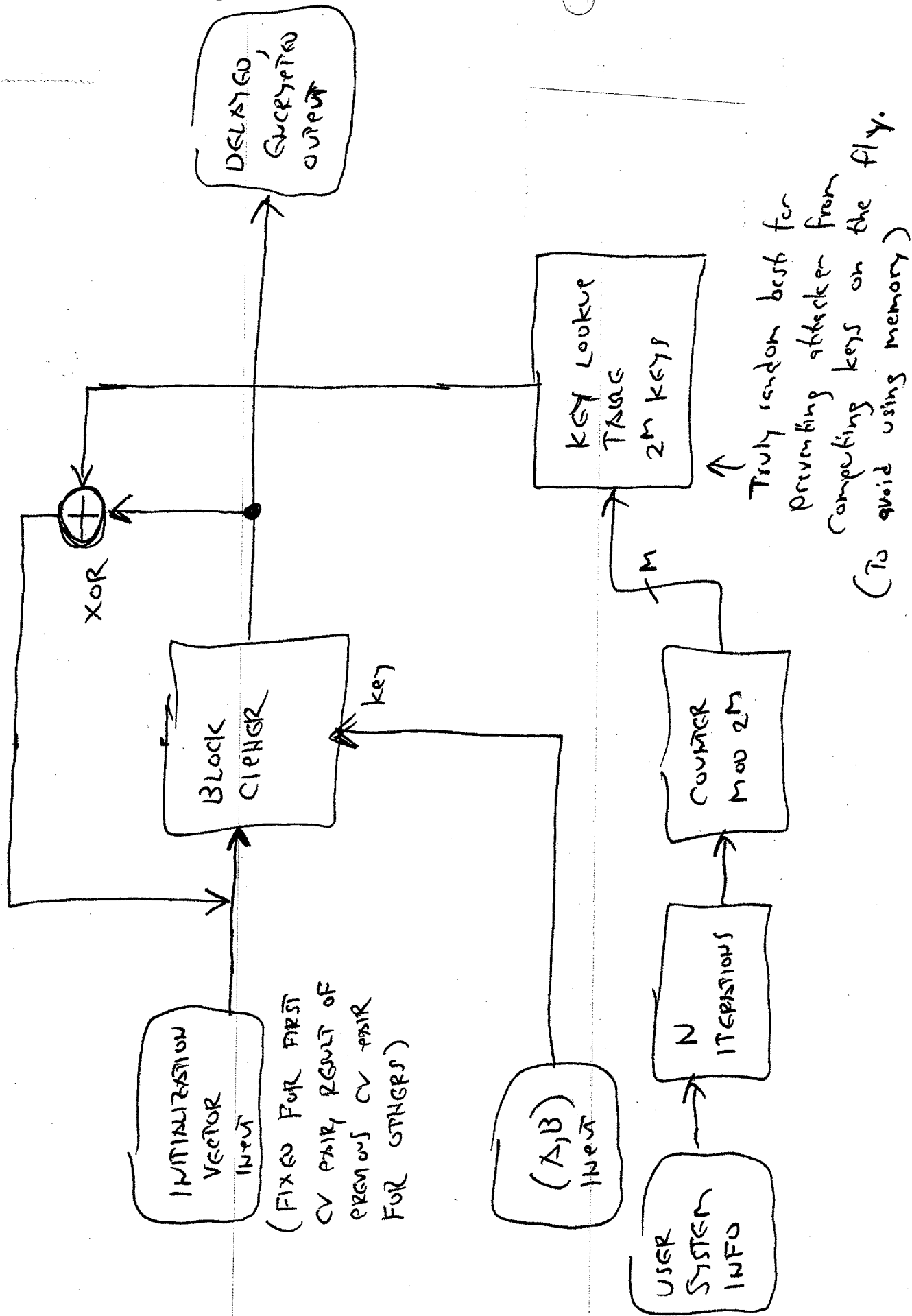
Private Key Delayed Unlock



Journal of Interpersonal Violence

Iterate





Number of consonants	13	
Number of vowels	5	
Combinations in each CV pair	65	
Pairs	8	
Total Combinations	318,644,812,890,625	(bits)
Base-2 Entropy	48	

1/2

<u>Mean Input Times</u> (experiment)		
Touch typing (fast), hidden digits	4.70 (sec.)	$= (5.3 + 4 + 5 + 5.1 + 4.1) / 5$
Tough typing (fast), digits shown	3.88	$= (3.8 + 3.5 + 3.4 + 3.5 + 5.2) / 5$
Mouse, drag line through digits	9.32	$= (9.7 + 9.2 + 9.8 + 8.9 + 9) / 5$
Mouse, click on digits	8.28	$= (8 + 8.2 + 8.9 + 8 + 8.3) / 5$
Set total delay to minimum total input time (Keeps user from noticing the delay)	3.88 (sec.)	

Software (equivalent machine)

Attack Analysis

Total number of seconds for all delayed combinations (on equivalent machine) 1,236,341,874,015,620

Average number of years on equivalent machine (1/2 total) 19,602,072

← Impossible with present machines

Effective lifetime of signing key (years) 20

Performance multiplier at end of life (Moore's law) 10,321

Total number of seconds for all delayed combinations (on future machine) 119,785,790,491

Number of future machines in network 1,000

Average number of years on future machine network (1/2 total) 0.95

← Even this number would leave evidence of fraudulent activity on the part of the person forging signature with broken private key

Massively Parallel Hardware (FPGA, ASIC) Attack Analysis

Budget (current equivalent dollars) 1,000,000

Cost per FPGA or ASIC (with NRE) 400

Number of available parallel branches in budget 2,500

Number of parallel branches operating simultaneously 2,048

Performance multiplier of each branch over equivalent machine 100

Total performance multiplier over equivalent machine 204,800

Total number of seconds for all delayed combinations 6,036,825,557

Average number of years (1/2 total) 96

← Impractical with present H/W, even w/ large budget

Effective lifetime of signing key (years) 20

↑ Can Include sunset date in ACI after which all sigs. are invalid

Performance multiplier at end of life (Moore's law)	10,321
--	--------

Total number of seconds for all delayed combinations (on future hardware system) 584,892

Average number of days on future machine (1/2 total)	3.38
--	------

But, here's where the key lookup helps protect against such attacks...

Random keys in key lookup table	8,192
---------------------------------	-------

Size of each key (in bytes)	16
-----------------------------	----

Total memory for lookup table (bytes)	131,072
---------------------------------------	---------

Total fast SRAM memory for all branches (bytes)	268,435,456
---	-------------

Total MB of fast SRAM memory	262,144
-------------------------------------	----------------

Cost per MB of SRAM (current equivalent dollars)	10
--	----

Total cost of SRAM	2,621,440
(See budget above.)	

should
(must fit in 256K cache)
to ensure top
performance in
user's machine.
otherwise ratio
between

$$\left[\frac{\text{Attacker Delay}}{\text{User Delay}} \right]$$

'is
reduced.

Loos of
gates: \$\$

$\sim 8 \times 2$ gates
per byte, or

$$\sim 4.3 \times 10^9 \text{ gates}$$

or $\sim 2M$ gates
per branch

Expensive ASIC!

SECURE DELAY OF HASH OUTPUT

A secure delay according to various aspects of the present inventions can be applied in other areas than just passphrase security. For example, a hash value can be run through a secure delay to produce a smaller hash value that would be computationally infeasible to derive based on a birthday attack. In one conceived embodiment, a 160-bit hash value is repeatedly run through a secure delay for a predetermined number of iterations that, given a security selection, corresponds to an acceptable unit delay. (An example of an acceptable unit delay is 1 second.) At the end of each unit delay, a sub-hash is computed from the current output of the secure delay and displayed.

A person wishing to compare hash values can begin comparing a first group of digits corresponding to the first sub-hash after the first unit delay, and while the second unit delay is underway. When the person looks for the second group of digits to compare, the second unit delay (when optimally chosen) is already completed and the third unit delay is underway.

Thus, a securely delayed hash system according to various aspects of the inventions can provide a smaller hash value with the same security as the larger hash value from which it is derived. The loss in entropy in the smaller value is offset by the computational difficulty (from the secure delay) of obtaining the smaller value. An attacker wishing to find a larger hash value that produces the smaller hash value will need to run the secure delay, on average, $N2/2$ times with the secure delay computation for each iteration.

If T = delay time (on an equivalent processor as that of the legitimate user), then $T2 = T * N2 / 2$. If $T = 1$ second, and the required $T2 = 1,000,000$ CPU years, then the required $N2 \sim 2^{21}$, a much smaller value than, say, 2^{160} .

Since a hash value is not particularly sensitive, it can be sent freely over in secure networks. It is conceivable that an Internet site can be established for quickly computing smaller "sub-hashes" based on transmitted hash values through an open-source, standardized secure delay algorithm. However, it is more likely that the market will demand simplicity and standardization, and an average delay within the reach of the average desktop PC. (The remote-computation model may be useful for portable computers, though.)

###

Choices: $7 \times 5 = 35 = M$

Digits: $N = 8$

no duplicate digits

\therefore

$$X = M \cdot (M-1) \cdot (M-2) \cdot (M-3) \dots (M-N)$$

$$X = \frac{M!}{(M-N)!} = \frac{35!}{27!}$$

$$X = 6.67 \times 10^{14}$$

$$\log_2(X) = 49.2 \text{ bits of entropy}$$

With 0.5 sec. delay between digits,
 $d = 4 \text{ sec}$

$$\frac{4X}{(3600 \times 24 \times 365)} = 84,496,818 \text{ yrs (!)}$$

on comparable delay processor

W/ 2x every 18 mos, in

20 yrs: 8,187 yrs (!)

AB-1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	P
Q	R	S	T	U
V	W	X	Y	Z
1	2	3	4	5
6	7	8	9	0

Passphrase:

In a variation, entry system is built

into keypad of USB-connected hardware delay processor

Already-selected digits can be illuminated

A	B	C	D	E
F	G	H	I	J
K	L	M	N	P
Q	•	S	T	U
V	W	X	Y	Z
1	2	3	4	5
6	7	8	9	0

Passphrase:

2050ED-462600T

A	B	C	D	E
F	G	H	I	••
K	L	M	N	P
Q	•	S	T	U
V	W	X	Y	Z
1	2	3	4	5
6	7	8	9	0

Passphrase: RJ _

A	B	C	D	E
F	G	H	I	••
K	L	M	N	P
Q	•	S	T	U
V	W	X	Y	Z
1	•••	3	4	5
6	7	8	9	0

Passphrase: RJ2 _

AB-3

205000-2462600T

A	B	C	D	E
F	G	H	I	.•
K	L	M	N	P
Q	•	S	T	U
V	W	X	Y	Z
1	•••	3	4	5
6	:::	8	9	0

Passphrase: RJ27 _

A	B	C	D	E
F	G	H	I	.•
K	L	M	N	P
Q	•	S	T	U
V	W	:::	Y	Z
1	•••	3	4	5
6	:::	8	9	0

Passphrase: RJ27 X _

AB-4

A	B	C	D	E
F	G	H	I	••
K	L	M	N	P
⋮⋮	•	S	T	U
V	W	⋮⋮	Y	Z
1	•••	3	4	5
6	⋮⋮	8	9	0

Passphrase: R327 XQ_

A	B	C	D	E
F	G	⋮⋮	I	••
K	L	M	N	P
⋮⋮	•	S	T	U
V	W	⋮⋮	Y	Z
1	•••	3	4	5
6	⋮⋮	8	9	0

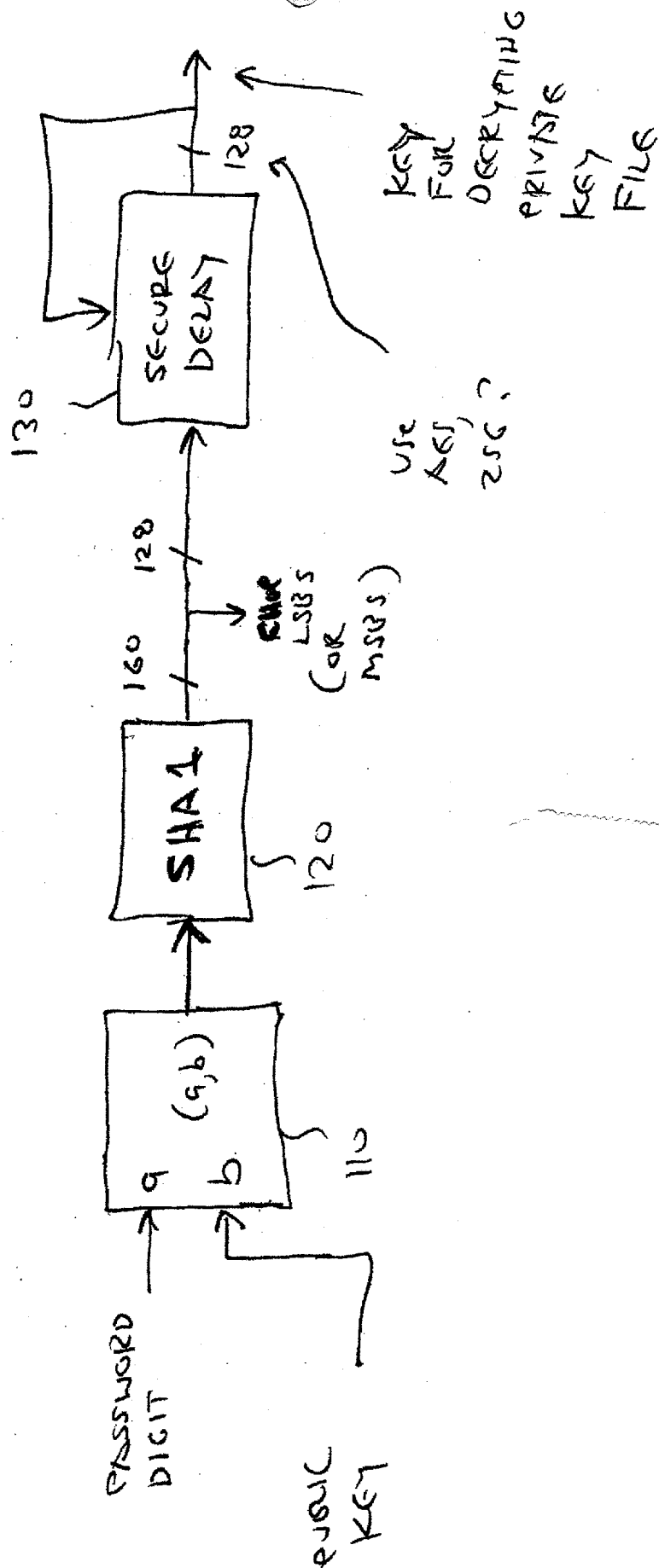
Passphrase: R327 XQH_

2050ED"EH52600T

A	B	C	D	E
F	G	⋮⋮⋮	I	⋅⋅
K	L	M	N	P
⋮⋮	⋅	S	T	U
V	W	⋮⋮⋮	Y	Z
1	⋅⋅⋅	⋮⋮⋮	4	5
6	⋮⋮	8	9	0

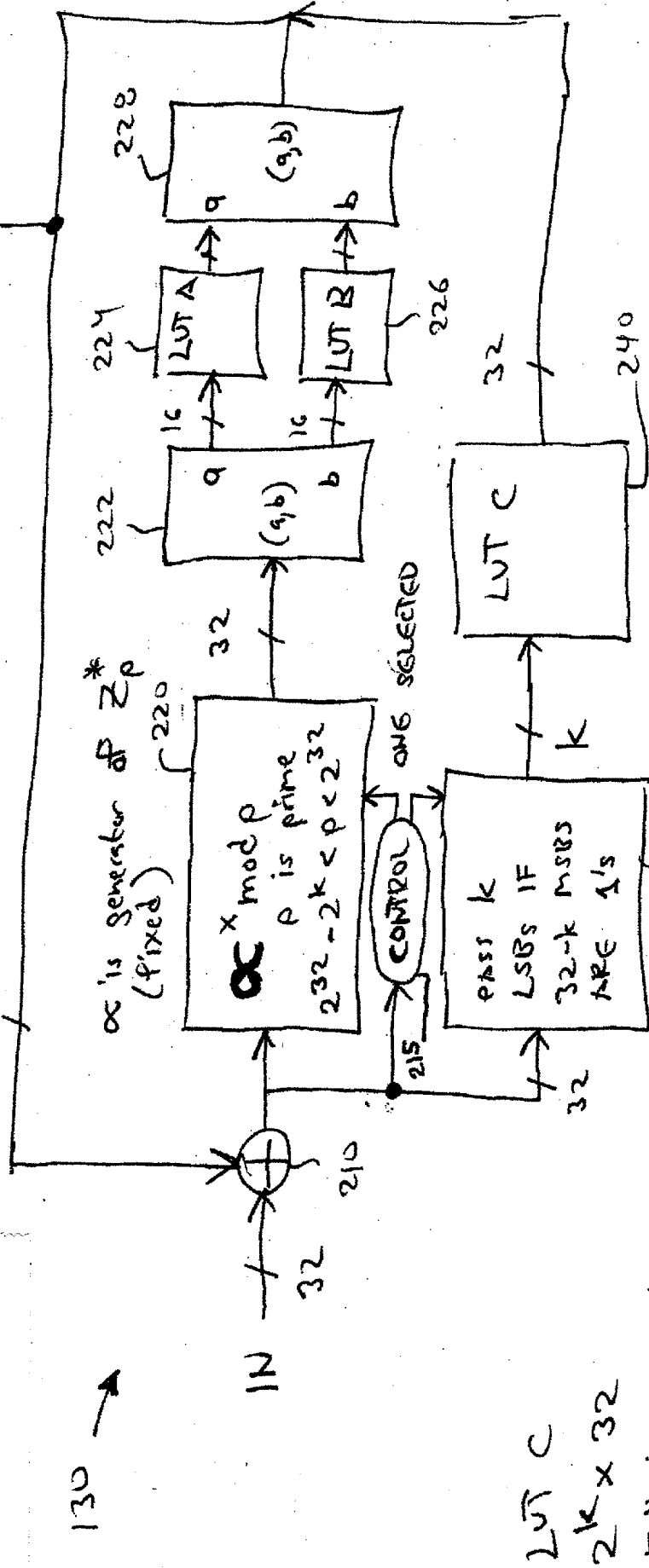
Passphrase: RJ27 XQH3

AB-6



2050ED E162600 AFTER N

ITERATIONS



LUT A, B 64K x 16

Can be same random mappings of $\{0,1\}^{16}$ or different.

ONE EXEMPLARY
EMBODIMENT OF
32-BIT SECURE DELAY

LUT C
 $2^k \times 32$

Fills in gaps in

$\{0,1\}^{32}$ not reached

by LUTs (A,B) because $p < 2^{32}$

$\alpha^x \bmod p$ not selected for $x \geq 2^{32-2^k}$.

ASSUMED

THERE IS A PROBABLE

PRIME $< 2^{32}$ and

within 2^k of 2^{32}

k small (4-5)

VIRTUAL SIGNATURE PRINTER

According to various aspects of the present inventions, an electronic record (e.g., MS Word document, AUTOCAD drawing, etc.) can be signed by "printing" that record to a special "virtual signature printer." An example of a virtual printer is the "PDF Writer" printer driver that is installed with the ADOBE ACROBAT software. The inventive "virtual signature printer" provides the user with an intuitive, simple way of authenticating an electronic record.

The "virtual signature printer" system produces an output file (or multiple files, see below) that can be sent to a recipient for viewing, printing, and validation. The recipient can view or print the file (preferably, the file is "backward compatible" compatible with widely available viewing software) and, with special software, can validate the signature on the file.

A particularly advantageous way of signing an electronic record that has been "printed" this way is with embedded signatures. However, a "virtual signature printer" system can employ any suitable technique for signing an electronic record. The following are some examples of output of such a system:

- A PS or TIFF file (or, with suitable licensing if necessary, a PDF file) representing the document, accompanied by a detached PGP-compatible signature file.
- A ZIP file containing a PS, TIFF, or PDF file representing the document including a ZIP comment containing a Base-64 PGP-compatible signature.
- A PGP-signed file containing the document in a PS, TIFF, or PDF file.

When the signer wishes to electronically sign a document he or she can print the document to the virtual signature printer driver, using the print functionality of the software used to create the document. The printer driver creates a window in which the software requests the signer's authenticating information. The user can enter his or her passphrase, apply his or her fingerprint to a fingerprint scanner, insert a smart card, etc. The software then computes the digital signature for the document, based on the authenticating information or a private key unlocked by the authenticating information, and embeds the digital signature with an output file or includes the digital signature in a separate file.

###

EDWIN A. SUOMINEN

U.S. Patents
5,926,513; 5,937,341*;
6,052,748*; 6,069,913
Additional patents pending*

14614 North Kierland Boulevard,
Suite 300
Scottsdale, Arizona 85254

Registered Patent Agent
Independent Inventor of Electrical
Engineering Technology

*Available for licensing

Telephone: (480) 948-3295
Facsimile: (480) 948-3387

www.eepatents.com
ed@eepatents.com

VIA EMAIL

Thomas E. Workman, Jr., Esq.
Law Offices of Thomas Workman
41 Harrison Street
Taunton, MA 02780

EXAMPLE

Re: Cryptography Technology

Dear Ted:

This brief disclosure is an example of a clear-signed document with a simulated digital signature (25x13 bits = 325 bits) that resides as a graphic within a signature block that is excluded from the signature calculation of the document. In an actual implementation, the user will place the signature block in the document and will then "print" the document to the digital signature printer driver. The driver will then create a graphic file such as a TIFF (multi-page if necessary), remove the graphics in the region where the signature block will go (setting that graphic data to a default blank value), compute signature for the entire document except the signature region, and place the signature data in the region as graphic-mode text.

The document's signature can be verified simply by opening it with a customized TIFF reader, which will detect the presence of the signature region and will validate the signature within it against the data of the document except the graphics within the signature block. An option can be provided to put the signature data on an entirely separate page of the document (e.g., after the last page), preferably with a facsimile copy of the signer's ACI. (In such embodiments, the ACI should have a blank space for the signature data of document signed with the ACI's signing key.) (AE-4)

An exemplary process is illustrated in the attached FIG. , in which a signer creates a document (e.g., this letter) using whatever software he (or she) wishes to use (e.g., Microsoft Word 97). He then prints the document to the SelfCertify.com virtual TIFF printer (call it a "software signature machine"?). The printer driver software creates a TIFF file of the document and displays it in a viewer window. The user interface of the viewer window requests the following input from the signer:

1. Selection of a graphical region within the displayed document for application of the signer's digital signature "stamp." The user can specify the region by moving a dashed box around the screen. The dashed box can have left and right arrows within it for navigating to different pages of a multi-page document, and an "Sign Here" or "OK" button for applying the digital signature "stamp" at the

AE-1

Thomas E. Workman, Jr., Esq.

Page 2

current location of the box. An unsigned signature page of this letter is attached showing what such a box might look like as it is moved around during the selection process.

2. Authentication from the signer to apply his digital signature to the document within the digital signature "stamp" at the selected location. As discussed in other disclosures, the authentication can be a securely delayed passphrase input.

When the signer has selected the location of the digital signature stamp and provided authentication for its creation, the printer driver software removes all graphic information from the selected location. It then computes the digital signature based on (1) the remaining data of the TIFF file (exclusive of the location of the stamp) and (2) the signer's private key.

Advantageously, the signed document is in a conventional format (e.g., multi-page TIFF) that can be read by any conventional viewer is signature authentication is not needed. When signature authentication is needed, a document can be viewed using the SelfCertify.com TIFF viewer (which may be distributed freely to encourage use of SelfCertify.com's digital signature services).

To verify a digital signature, the viewer searches the graphical rendering of the signed document for the distinctive graphical outline of the signature block. Distinctive features of the graphical outline can include (1) a distinctive color such as maroon, (2) a distinctive line shape such as double parallel lines, and (3) a distinctive line weight such as 3.2 points (not an integer or x.5 fraction). All of these distinctive properties are present in the simulated signature block for this document. In addition, the signature block can have a predetermined size to further identify it.

Once the viewer software has identified the exact location of the signature block, it removes all of the graphic data of the signature block (up to and including the border) and sets it to the default blank value used during signature calculation. It performs an optical character recognition of the signature data within the block to obtain the value of the digital signature. According to advantageous aspects of the invention disclosed elsewhere, the digital signature can be applied to a secure delay to obtain another, larger digital signature value. The digital signature value (as shown or as expanded through a secure delay) is then validated against the modified TIFF representation of the document.

Very truly yours,

NOTE: Signature data can be in the form of a bar code (1D or 2D). If license to PDF format could be obtained, could include signature characters as data (not image pixels) and search would be very simple search for key tag characters in file.

-SelfCertify.com Digital Signature-
1410-2708-9007-5781-6673
7701-3376-8745-3789-9100
7440-2788-7101-4089-2409
5420-2178-9001-7680-1477
7731-3376-2645-3789-9105

Edwin A. Suominen

AE-2

Thomas E. Workman, Jr., Esq.

Page 2

current location of the box. An unsigned signature page of this letter is attached showing what such a box might look like as it is moved around during the selection process.

2. Authentication from the signer to apply his digital signature to the document within the digital signature "stamp" at the selected location. As discussed in other disclosures, the authentication can be a securely delayed passphrase input.

When the signer has selected the location of the digital signature stamp and provided authentication for its creation, the printer driver software removes all graphic information from the selected location. It then computes the digital signature based on (1) the remaining data of the TIFF file (exclusive of the location of the stamp) and (2) the signer's private key.

Advantageously, the signed document is in a conventional format (e.g., multi-page TIFF) that can be read by any conventional viewer is signature authentication is not needed. When signature authentication is needed, a document can be viewed using the SelfCertify.com TIFF viewer (which may be distributed freely to encourage use of SelfCertify.com's digital signature services).

To verify a digital signature, the viewer searches the graphical rendering of the signed document for the distinctive graphical outline of the signature block. Distinctive features of the graphical outline can include (1) a distinctive color such as maroon, (2) a distinctive line shape such as double parallel lines, and (3) a distinctive line weight such as 3.2 points (not an integer or x.5 fr. properties are present in the simulated signature block for this document. In addition, the signature block can have a predetermined size to further define it.

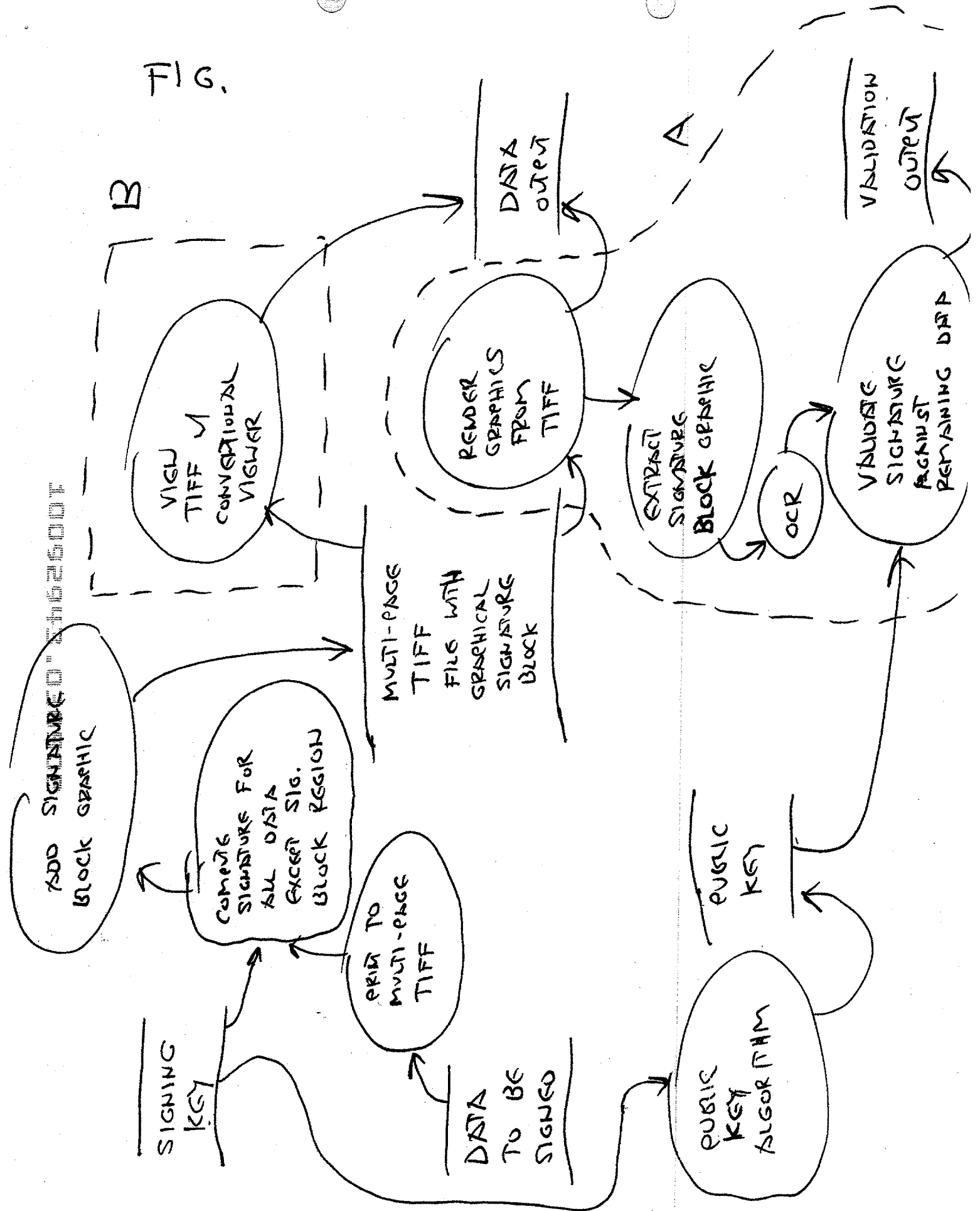
Once the viewer software has identified the exact location of the signature block, it removes all of the graphic data of the signature block (including the border) and sets it to the default blank value used during signature calculation. It performs an optical character recognition of the signature data within the block to obtain the value of the digital signature. According to advantageous aspects of the invention disclosed elsewhere, the digital signature can be applied to a secure delay to obtain another, larger digital signature value. The digital signature value (as shown or as expanded through a secure delay) is then validated against the modified TIFF representation of the document.

Very truly yours,

Edwin A. Suominen

AE-3

FIG.



ALTERNATIVE: Conventional TIFF + detached

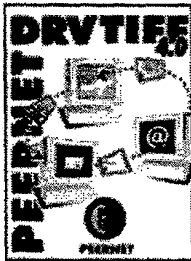
PGP signature
(e.g. in Zip
file)

PEERNET.DRV TIFF Driver

[Selecting a Driver](#) | [Supported Platforms](#) | [Common Features](#) | [Support](#)
[Online Solutions](#)



PEERNET



4.0 online
MANUAL
14 day
TRIAL 4.0
3.0 online
MANUAL
14 day
TRIAL 3.0



Enterprise
Labeling Suite
Raster Print
Driver Products

Free Trials

The Store

Contact Us

Home

PEERNET.DRV TIFF Driver converts any document capable of being printed by a Windows application into high quality serialized or multi-page TIFF images. It is ideal for imaging or archiving applications. It's also a handy file-generation tool for cross-platform article distribution.

TIFF conversion is as fast as printing. Document scanning is obsolete. Paper waste is a thing of the past.

Features of PEERNET.DRV TIFF Driver

Append Mode (Version 4 only)

Build a multi-page file gradually by appending pages to an existing file whenever the need arises.

Color Correction

Lets you adjust the image's appearance to compensate for monitor non-linearity on Windows NT 4.0 and Windows 2000.

Color Modes

Color or "Black and White only".

Color Reduction

Turn automatic color reduction on or off, to suit your needs. Automatic color reduction reduces your image to its fewest number of colors without affecting picture quality. In addition, automatic color reduction selects the optimal output format for serialized files or the optimal output page format for multi-page files.

Compression

Turn compression on or off as needed.

Output Formats

TIFF True Color Uncompressed or Compressed Packbits
TIFF Monochrome Uncompressed or Compressed CCITT Group 4
TIFF 256 Color Uncompressed or Compressed Packbits

Paper Size

Converts custom documents up to 18.03 x 18.03 inches, or 458 x 458 millimeters. Supports most paper sizes.

Resolutions

100, 200, and 300 DPI resolutions.

Can use
existing
technology
to
print signed
documents
According to
various aspects
of the present
inventions.

AE-5

ZDNet DOWNLOADS

- Top 100 Products
- Download The Future
- Free Downloads

ZDNet > Downloads > Utilities > Printer Utilities > **EZ-Printer for Windows NT**



Search For:

PC Downloads

GO

- Search Tips
- Power Search

Downloads Home

Pick a Category:

Games
Internet
Utilities
Home & Education
Small Business
Graphics & Multimedia
Development Tools
Windows 95/98/NT
Commercial Demos

See Also:

Editors' Picks
ZDNet Exclusives
PCMagazine Utilities

More Downloads:

Macintosh
Windows CE
PalmPilot
LaunchPad



FREE computer book with membership

Services:

Help
Upload a File
Free Newsletter
SmartPlanet

EZ-Printer for Windows NT

10-26-99

Print to image file from any application

ZDNet Rating ★★★★★ User Rating NR

Be One of the First to Rate This Title!

Company: Suntop
Version:
Size: 774.95 KB
Downloads: 485

Download Now

Get **GO!ZILLA**
And Download Better

- ▶ Add to Basket
- ▶ Look in Basket

- View file contents

Requirements: Windows NT 4.0

Purchase Information: Demo: \$49-\$59 for retail version.

EZ-Printer for Windows NT makes it possible to print from any application to an image file. It installs itself as a native device in the printer control panel. Users can simply choose Print and select the EZ-Printer printer from the list of available printers. Output is in black and white (the author has a color version, too) and is automatically saved in the file format of choice (DIB BMP, GIF, TIFF, or PNG). The images are automatically sequentially numbered. The included viewer will let users see the results immediately. This demo version of the driver includes a banner in the middle of the image. Virtually no documentation is included, so users will need to visit the authors' Website to get information or support. Output also seems to be limited in resolution, with no apparent way to control the dpi. Still, EZ-Printer could be particularly useful for Websites, writing documentation, presentations, and many other applications.

Reviewed on Oct 23 1999.

Download Now

Add to Basket



See Related Links

Hot Clicks

- Yahoo! Messenger
- Today's Free File
- Instant Software
- Weekly Top 20

SPECIAL OFFERS from our Partners

Faster Downloads!

It's Free! The Go! Zilla download manager

One-Click Answers

Get GuruNet for simple, fast, relevant information.

TaxACT 2000 Preview

Get a jump start on your 2000 tax return!

Fix your PC for FREE!

Optimize Windows with System Mechanic-try it free!

Protect Your PC

Download Sygate Personal Firewall. It's FREE!



i-drive Sideload

E-mail this



Print this

Another example of existing TIFF generation technology that may be adapted for use according to various aspects of the present invention.

AG-6

PRINT TO SIGNED TIFF FILE

Appendix AF

According to various aspects of the present inventions, a document can be signed by printing the document to a TIFF file using a virtual printer driver, e.g. provided by a service such as "SelfCertify.com." (Note that SelfCertify.com is not an operating business entity, though the applicant has registered the domain name, and has not offered any product or service for sale as of the filing date of the present provisional application.) The TIFF file is created as it normally would be except that includes a signature block within a suitable field.

In one embodiment, the signature block is included within the TIFF "ImageDescription" field, the ASCII contents of which are excluded from the signature calculation of the file. (See AF-3.) In an actual implementation, the user "prints" the document to the digital signature printer driver. The driver will create the TIFF (multi-page if necessary), with blank characters in the fixed-length "ImageDescription" field where the signature data is intended to reside (setting that data to a default blank ASCII value), compute signature for the entire document with the default blank value in the "ImageDescription" field, and place the signature data in the "ImageDescription" field as ASCII.

The document's signature can be verified simply by opening it with a customized TIFF reader, which extracts the signature data from the "ImageDescription" field and validate it against the data of the document with default blank values substituted in the "ImageDescription" field.

An exemplary process is illustrated in the attached FIG. (AF-4), in which a signer creates a document using whatever software he (or she) wishes to use (e.g., Microsoft Word 97). He or she then prints the document to the SelfCertify.com virtual TIFF printer. The printer driver software creates a TIFF file of the document and displays it in a viewer window. The user interface of the viewer window requests authentication from the signer to apply his digital signature to the document. The authentication can be a securely delayed passphrase input according to various aspects of the present inventions.

Advantageously, the signed document is in a conventional format (e.g., multi-page TIFF) that can be read by any conventional viewer is signature authentication is not needed. When signature authentication is needed, a document can be viewed using the SelfCertify.com TIFF viewer (which may be distributed freely to encourage use of SelfCertify.com's digital signature services).

The signer's public key can be included in the signature data along with the digital signature and the name of the signer. A facsimile copy of the key ACI (see, e.g., Appendix A) can be appended to the end of the TIFF file after the document pages.

AF-1

205000-0462601

Having both of these items present in the TIFF file permits a verifier to quickly authenticate the signed document. If he or she is satisfied with the integrity of the TIFF reader/verification software he or she has obtained (e.g., from a secure, trusted web page of SelfCertify.com), and if he or she is satisfied that the facsimile copy of the ACI with its security background digits is not a forgery, he will have everything he needs to validate the signature of a person who has not made any prior digital assigning arrangements with him.

Signing and verification software according to various aspects of the present inventions can be integrated with the GPG ("GNU Privacy Guard") software, which can be freely distributed and modified under the GNU Public License. The signing and verification software can call the GPG software with parameter passing. The inventor of signing and verification software can thus include, generally, a slightly modified TIFF printer driver and TIFF reader that calls the GPG software for all digital signature functionality. All three pieces of software can be released in a single compact package.

The TIFF reader software can provide the option to output the original TIFF file (without signature data in the "ImageDescription" field) along with a PGP-compatible detached signature to the file and an ASCII file with the signer's public key. This permits less trusting users to verify integrity of the signature and signing key (comparing PGP's SHA-1 "fingerprint" to what's shown on the ACI) with their own copy of PGP or GPG.

###

Sample "ImageDescription" Tiff Field

Tag = 270

Type = ASCII

Count = Fixed number of characters in signature block, plus 1 for NUL at end.

Value = 1728

"-- SELF CERTIFY.COM SIGNATURE DATA -- ↵

Signed by: Edwin A. Suominen ↵

mQGIBDk8ZlMRBADZZ0behMne0qwL7mu7fa/KfbPx2wLtMSihh3IitOo6o6e/twYQ ↵
3Z27YlIlu9uvhIkdsBrQ7b+N0paKyJAu691eE5gzP8VEdzfLJtCQDXvdO9+H57Er ↵
PGicVujuGGIPxvzA7QuyxNxDzndKtFIGO60zn452pWrg/77iA+Ne0CYCuQCg/9I3 ↵
6bNa0vfxXUV3CS+/PDo9VpED/jwquHOYJQYOi0jZNdaT9ZN8mzRQlPgfyGHuNBpp ↵
yoPRhck0nMe1BxNG83M2v243M0DUMabCPuGqqOtYKe5YLqAw/iM5IWwp3EctEHGU ↵
48r8gC7rYKRHOLosyBfx6/uQkpGiNeM4TAI9QpqVzsbHvF1H5LSauLbHOSAwtUoM ↵
e/+uBACTQjV15XA+MPwaIx1vZ3lD2iEX/XFL0edxA5XzcN9uVlet+Chxd7xJn66x ↵
w0xzdvLvb/kCdcNY8idJkJVqAUx249S+PymCQFR+sX0pxXCVky4JgtLDToX0wW1G ↵
6C0kPzUR1AH9jdAUaPc+7SC1TdixmOPsLR1l+5PzUUVhNj6b8bQgRUFTIEp1bmUg ↵
MjAwMCA8ZWRAZWVwYXRlbNzRmNvbT6JAFQEEBECABQFAjk8ZlMFCQAiRwAECwMC ↵
AQIZAQAQKRCRZ8BksUXUY2Y3AKCG99iXRgxGmOssyOC0Lwm/U0yECACfW6R9rI2f ↵
G+UeNOWE/b2TJdt49La5Ag0EOTxmUxAIAPZCV7cIfwgXcqK6lqlC8wXo+VMROU+2 ↵
8W65Szzg2gGnVqMU6Y9AVfPQB8bLQ6mUrfdMZIZJ+AyDvWXpF9Sh01D49Vlf3HZS ↵
Tz09jdvOmeFXklN/biudeE/F/Ha8g8VHMGHOFmlm/xX5u/2RXscBqtNbn02gpxI6 ↵
1Brwv0YAWCv19Ij9WE5J280gtJ3kkQc2azNsOA1FHQ98iLMcfFstjvbySPAQ/Cl ↵
WxiNjrtVjLhdONM0/XwXV00jHRhs3jMhLLUq/zzhsS1AGBGNfISnCNLWhsQDGCgH ↵
KXrKlQzZlp+r0ApQmwJG0wg9ZqRdQZ+cfL2JSyIZJrqrol7DVeKyCzsAAgIH/0V8 ↵
DY5pj51RDGsakRhMebL9Ob7v9GsbZN6PfTg02upuCi6WUyazabw4J4ZFc7vtpo8x ↵
FQOkCofOLmisNim7r0PyWrW0SgHLbcXwMMUUb1h/QbggH0WtkkJTzXgNGL+MLJZa ↵
ND4R0gle03PQep4SZgA6/x9OUGWStmzWEt3jk/VdnImS5gDjMNHmCX7+ZaCxROiL ↵
zO3oDmzIRpVYk3+tnekDVhhrDwX5lQ1zUoCg43hAmA1Q1/KNFBw/qiol0EvLyJby ↵
hUzhGqdzd/MJkNHXviOoJyuOnQH+O81EME5S2Ej19epf4Rfuf9rn8uR7t13YERaD ↵
wqwO4VICd5n+6F3199GJAEWEGBECAAwFAjk8ZlMFCQAiRwAACgkQkWFaZLFF1GMG ↵
KwCZARTQgJDOM40GBp00JwPlEscVP/gAoPIJb/gmbNpbeQmG9UobWiT8PKl1 ↵
=zFrB ↵

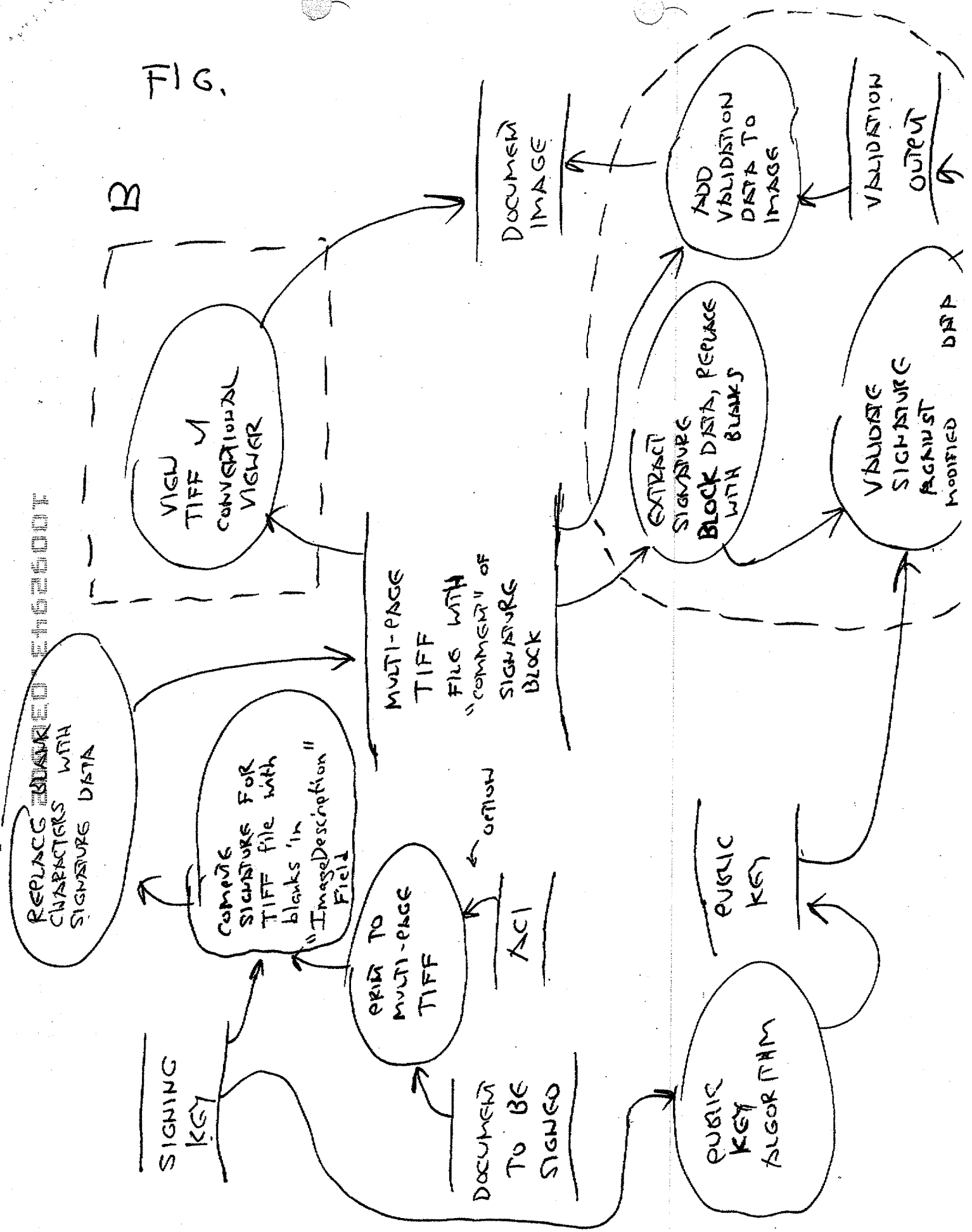
↵

Signature: ↵

iQA/AwUAOhs1jKmKuMvNCWDGEQJ4TACeJpwTCOzNvxKhZVYag17lBEuhKEMAnjtT ↵
SivKAZgC21P/pMrro2HgTfJo ↵
=Adug<NUL>"

FIG.

B



IMPROVEMENT TO PGP
By Edwin A. Suominen

Allowing keys of keyring to exist at different locations

I would like to follow the advice of some encryption experts to keep my private key (or at least a signing key) on removable media (e.g., a floppy disk) for added security. However, the only way to do this now is to have my entire PGP keyring on that floppy disk. If the key I'm truly worried about is to be Secure, I won't want to keep that floppy disk where it is easily accessible. Instead, I will want to keep my public keys and my encryption key on the hard disk and just my signing private keying on a floppy disk.

According to this proposed improvement, PGPkeys allows users to specify the file location of private keys on their keyring. Everything works normally except when an "indirect" private key is to be used, e.g., to apply an especially important PGP signature to a contract. When that happens, the PGP software looks for the file containing the indirect key. If the file is located on removable media that is not in the drive, the software prompts the user to insert the removable media containing the indirect key. If the file is located on a PGP disk or some other type of encrypted volume, the software could simply indicate that the volume is not present and invite the user to mount it.

After use of the PGP key is complete, the software can advise the user that it is no longer needed and that he or she can unmount the volume or put the removable media back in its secure location.

1005043-030502

ed@eepatents.com, No Subject

To:
From: Ed Suominen <ed@eepatents.com>
Subject:
Cc:
Bcc: ed@eepatents.com
Attached:

LAW OFFICES OF LOUIS J. HOFFMAN, P.C.
14614 North Kierland Boulevard, Suite 300 * Scottsdale, Arizona 85254
Telephone: (480) 948-3295 * Facsimile: (480) 948-3387

Edwin A. Suominen * Admitted to practice in patent matters before the U.S. Patent Office only
Web Site: <http://eepatents.com> * PGP Public Key: <http://eepatents.com/key>

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

This is an example of a message that has been signed with **preserved formatting** and an *unobtrusive* digital signature around clear-signed text, according to various aspects of the present inventions.

-----BEGIN PGP SIGNATURE-----

Version: PGP Personal Privacy 6.5.8

iQA/AwUBOqO3eqmKuMvNCWDGEQLwPwCePjly0iuPEKIErsSyqTCA7S++MpIANRPv
qtttmsePjh/WqGafymg/hVMs
=q4Oy

-----END PGP SIGNATURE-----

CRYPTOGRAPHIC DOCUMENT DESTRUCTION

BACKGROUND AND SUMMARY

In private confidential conversation, two people can have a conversation without leaving any record of their conversation. With written or electronic communications, however, there is some record of what was said. That record can be difficult to eliminate.

Paper communications such as letters can be shredded if both sender and recipient agree that they will destroy their copies. Electronic communications (e-mail) are more difficult to eliminate because backup copies can be made and automatically archived onto other locations. It is sometimes surprising that backup copies are available during discovery of communications that would be embarrassing. Accordingly, there is a need for a system of destroying electronic communications or records when the sender and recipient of the communications agree to do so.

A system according to various aspects of the invention includes: an encryption subsystem; and a decryption subsystem, the decryption subsystem using a temporary key that can be disposed of to make encrypted communications unreadable.

DETAILED DESCRIPTION

An encryption key allows an authorized person to decrypt encrypted communications. For example, an encryption key can be a passphrase, or use a passphrase, known only to a person authorized to decrypt communications. According to various aspects of the present inventions, the decryption key can be destroyed. A passphrase for such a decryption key is preferably forgettable, for example a random alphanumeric string of sufficient length to be secure. Advantageously, the alphanumeric string can be used as a passphrase to open communications or records when is desired to do so and then destroyed and forgotten about when it is no longer desired for such communications or records to ever be decrypted again.

Operation of one embodiment includes (1) writing an electronic mail message to a person; (2) encrypting the communications using an agreed-upon passphrase, preferably an alphanumeric random digit string, for example 12 digits in length; (3) sending the encrypted message to the recipient; (4) having the recipient type in the passphrase to open the encrypted communications; and then after a predetermined or agreed-upon period of time, (5) having both parties destroy the passphrase (throwing away a Post-It note upon which the passphrase is written) so that neither the sender nor the recipient can ever decrypt the communication again. Preferably, the passphrase is used only for a short length of time or limited number of times so that it is impossible

for either party to remember it. The more random and arbitrary cryptic the passphrase is, the more difficult it will be to ever remember.

Systems according to various aspects of the invention can be useful in the legal profession where sometimes legal professionals are called upon to testify about matters that were assumed to be privileged but the court determines that they are not for whatever reason, as happens in patent practice sometimes. If an attorney or agent has communicated with his client using this system, and the client agrees to destroy the passphrase after the matter is complete, and the device communicated by the attorney or agent is no longer relevant or needed and has been acted upon completely, then it is impossible for any court or any party to discover what to parties discussed.

Even if a backup copy of an electronic mail message is found, a court can authorize a cryptanalysis of the message, but if it is encrypted using PGP strong encryption, it would be very difficult, effectively impossible, for the opposing party to figure out what the message said.

Embodiments can be employed in other types of communications that are encoded in digital form so that they can be encrypted. Even handwritten notes can be scanned into digital form and encrypted.

Voice messages can be digitized and compressed, entire paper files can be archived by scanning and digitizing, and then encrypting into a single encrypted archive file with a temporary key that can be disposed of after a predetermined time, which can be set by policy, for example one year.

According to another aspect of the invention, the keys need not be remembered or typed in by a human operator at all. According to this aspect, the key is an actual hardware device that transmits decrypting authorization indicia for a predetermined or agreed-upon period of time and then is incapable of doing so after that. An example of such a key is placed between a conventional PC keyboard and a PC. The device includes circuitry for reading a decryption key code or indicia from another device such as a card having barcodes printed on it or a disposable integrated circuit, which can be made in the form of a key.

The device can be sold with a number of keys that can be used and disposed of by the user. For example, if the device is sold with twelve keys with refills of 12 additional keys available by ordering, the user can encrypted archive records every month with a different key and cost a new (different) key away every month. The user may wish to keep the records on file for a period of several months in which case the user will begin using a key one month and then put the key into storage for a couple of months and then toss the key away, destroying it irretrievably after that period of time.

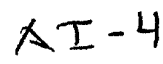
An embodiment using printed cards is less expensive and the keys can be disposed of more cheaply but it is more prone to unauthorized or inadvertent duplication, in which case the whole purpose of the system might be defeated. The user of such a system needs to take precautions that the keys are never duplicated.

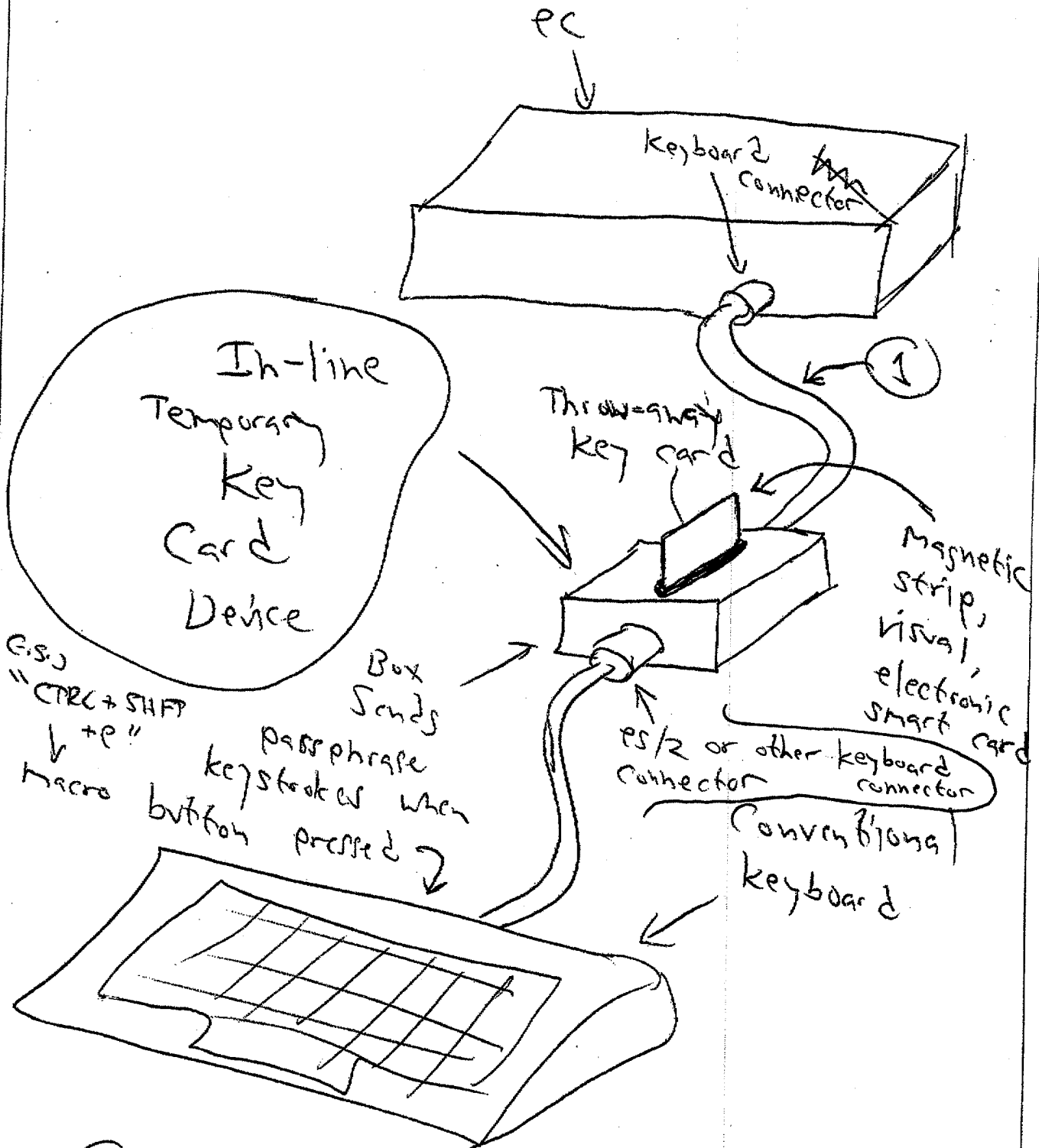
A database of which files correspond to which temporary keys can be created according to aspects of the invention so that an administrator can look over the list of keys about to expire and ask the persons involved with the effective files whether or not they need information from the files before they are destroyed. Paper documents can be shredded at the same time the keys are destroyed. If the key for a paper file that has a corresponding electronic file is a card, the card can be kept with the paper file and both can be destroyed simultaneously.

The key can be distributed from a sender of information to a recipient who is only authorized to access the information for a temporary period of time, for example one or two days. The sender of the information, or provider, can demand the key back after that period of time. In such a system, the key needs to be difficult to duplicate, for example an integrated circuit in the shape of a key. A forgettable password would not work for such a system because the user could write it down without telling the sender, but the forgettable password system works well when both parties, or all parties involved, are in cooperation and consent to destroy the information and the forgettable password.

###

INFORMATION





① Can be tamper alarmed, won't send keystrokes if ~~tap~~ is detected
pass-phrase ("key")

13-752	500 SHEETS FILLER	5 SQUARE
42-381	150 SHEETS EYE-EASE®	5 SQUARE
42-382	100 SHEETS EYE-EASE®	5 SQUARE
42-383	200 SHEETS EYE-EASE®	5 SQUARE
42-382	100 RECYCLED WHITE	5 SQUARE
42-389	200 RECYCLED WHITE	5 SQUARE

also
(can be used with permanent key systems)

Physical "Private Key"
for conventional
crypto. systems
(e.g., PGP, hushmail)

Or magnetic
card

In-line
box,
pass phrase
stored on
key, sent
when key combo
typed on

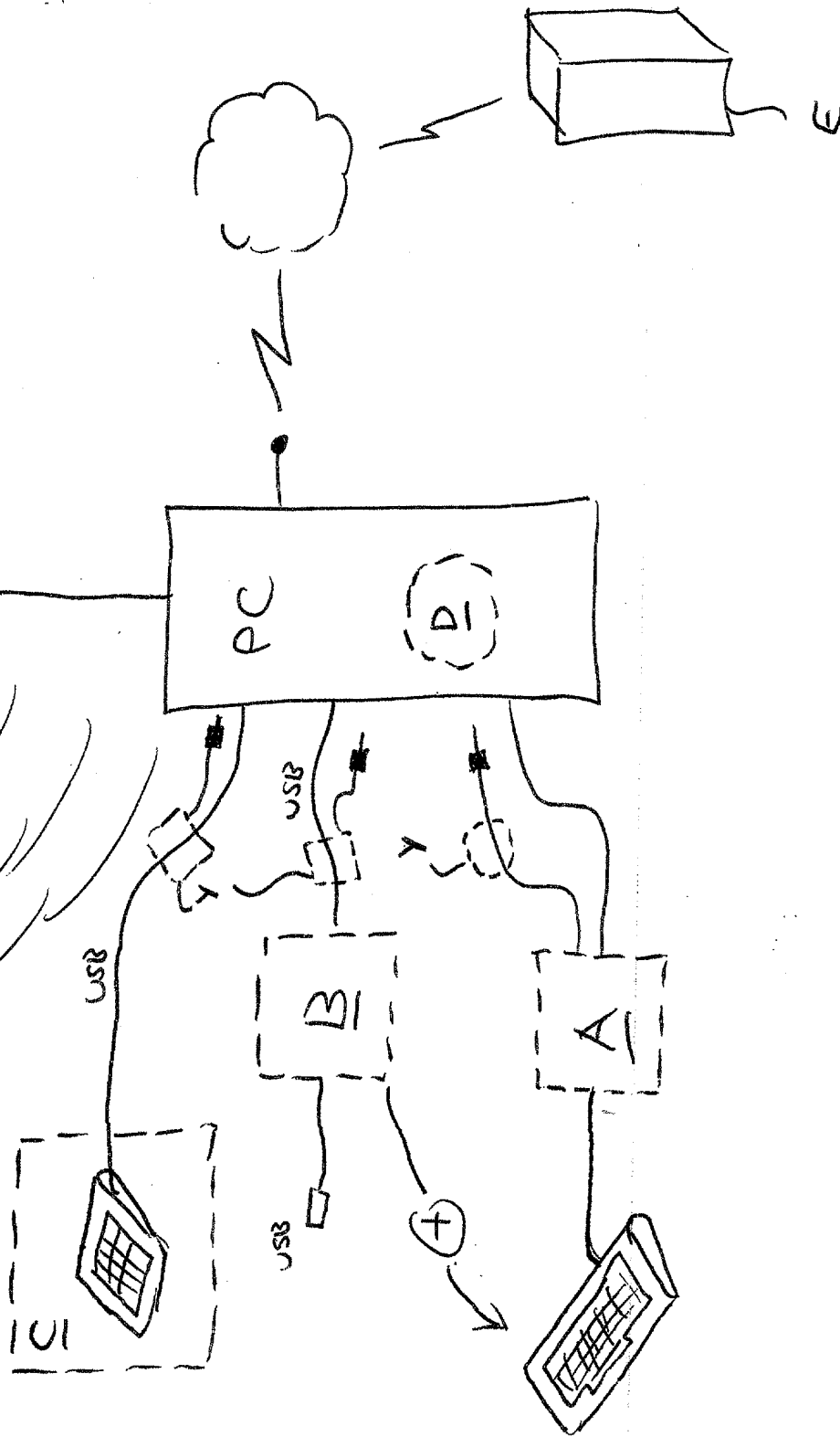

Conventional
Keyboard

* or fingerprint
(or other biometric)
Sensor w/ "patterns" as
"passphrase" or "key"

Or could hang off port, e.g. USB

(but then raises question of shifting ports)

1007
X



CLOCK JITTER RANDOM BIT GENERATOR

This invention uses existing hardware in a standard desktop PC with some very compact software to efficiently generate truly random numbers. Tell that introductory sentence to any cryptographer and they will either (1) beg you tell them how, or (2) dismiss you as an ignorant crackpot and lead you to various references talking about detecting hard disk read latencies, digitizing thermal noise, counting bubbles in fish tanks or lava lamps for generating true random numbers. (Pentium III chips have a hardware random number generator embedded in them, but I'm not sure that the AMD chips do. In any event, this invention provides random numbers from hardware that exists in all PCs today, and the need is still out there for a simple software solution.) I think if they were to read the following paragraphs they would scratch their head and say "Now why didn't I think of that?"

A method of generating a random binary bit according to various aspects of this invention includes: (a) clocking a CPU using a phase-locked loop; (b) beginning a count of CPU clock cycles upon a first transition of an independent clock signal (e.g., the PC's real-time clock, which generates interrupts on IRQ08 under control of a 32.768 kHz crystal); (c) recording the number of CPU clock cycles counted upon a second transition of the independent clock signal; and (d) extracting the least significant bit of the count to serve as a random binary bit.

Modern microprocessors use high-frequency clocks that are generated using a phase-locked loop (PLL). The PLL multiplies a lower-frequency crystal oscillator clock signal to generate microprocessor clock, as well as other clock signals used in the system. The Intel "CK98 Clock Synthesizer/Driver" is an example of a chip containing a PLL for this purpose.

PLL's lock the output frequency of a relatively unstable voltage-controlled oscillator to the more stable reference frequency of a crystal oscillator. The locking is performed using a feedback loop that has particular characteristics. Because phase locking cannot be perfect, there is always some "phase noise" or jitter on the PLL output. This phase noise or jitter causes the frequency of the PLL output signal to randomly vary within a Gaussian distribution. CPU clock generators are not designed to have very low phase noise because the exact clock frequency of the CPU is not particularly important. (That is in contrast to the type of PLL design I'm familiar with, for cellphone receivers, where the phase noise needs to drop off within a few hundred Hz of the carrier.) The PLL of the CK98 chip, for example, specifies that the "ideal closed loop jitter bandwidth" be attenuated 20 DB at 500 kHz, and a graph in the chip's data sheet shows the jitter (phase noise) spectrum

extending out to 50 kHz without significant attenuation.

with these levels of jitter, the number of CPU clock cycles in any given time interval can be expected to have significant variance. The high frequency components of the phase noise will integrate over long intervals and cancel each other out to some extent, but the number of CPU clock cycles over one predetermined time interval will be different (in a random fashion) from the number of CPU clock cycles over another predetermined time interval of the same length.

As a time interval gets longer, the number of clock cycles over the interval increases and the possibility of an integer difference between the number of clock cycles in two separate intervals of the same length increases proportionately. However, longer intervals permit high frequency variations in instantaneous frequency to cancel each other out.

The probability of any given frequency measured over a given time interval is defined as a Gaussian function of the frequency offset from the correct PLL frequency. Thus the probability of any given offset frequency measured over a larger time interval can be expected to be smaller than the probability of that frequency over the shorter time interval, by a scaling factor inversely proportional to the square root of the ratio between intervals. So the probability P of an offset frequency f_{dev} measured over an interval T is a function $P(f_{dev}) = P(f_{offset}) / \sqrt{T/C}$. If the interval is quadrupled, the probability of measuring a given offset frequency over that longer interval is 1/2 the probability of measuring that frequency over the shorter interval.

However, the number of clock cycles employed in the frequency measurement increases linearly with the length of the interval. The measured frequency deviation (over the interval) represented by a single LSB of difference in clock cycles is inversely proportional to the number of clock cycles. So, longer time intervals permit more accurate measurement of a given frequency deviation. If an interval is quadrupled, the probability of a given offset frequency difference resulting in an LSB difference is four times the probability at the shorter interval.

In view of the above two paragraphs, it is clear that any amount of random frequency deviation can be measured so long as the interval is made long enough. The probability of measuring a given offset will decrease, but the ability of the measurement to detect that offset will increase faster than the decrease in probability of offset. In addition to this rather crude analysis, an additional factor may be the presence of low-frequency deviations in the Gaussian distribution. It is possible that the integration of the area under the Gaussian curve from zero offset to $1/f_{dev}$ will increase to the point where a value of $f_{dev} = 1/T$ will become likely.

The existence of a PLL to control the CPU clock and a separate crystal oscillator to control a real-time clock in every PC (and probably Macintosh) out there makes this method an extremely useful way to generate truly random bits.

FAQ

Q1: What about other pseudorandom variations caused by other interrupts, which may slow down the count in a predictable fashion?

A1: Uncorrelated variances add, so the presence of any truly random component in a binary number will result in a random LSB as long as the truly random component is at least one LSB in size.

Q2: How large would the count of CPU clock cycles be with a Pentium III running at 500 MHz and a 32.768 kHz interrupt rate from the real-time clock?

A2: 15,258.8

Q3: Obviously, you can't measure 0.8 clock cycles. You probably can't measure integer clock cycles either because of the count routine requiring multiple clock cycles. So what sort of resolution could you expect in that count?

A3: Probably a count by 3's, or 5086 possible count values. One for the increment, one for the branch back, one for unknown possibilities.

Q4: So what is the minimum frequency deviation (in Hertz) you could measure over one 30.5 microsecond interrupt interval from the real-time clock?

A4: 98 kHz.

Q5: What if the frequency offset is unlikely to be that large over any given interval, say only a 10% probability?

A5: Accumulate 100 potentially random bits over 100 real-time clock intervals. The chance of every single one of those intervals *not* producing a random bit is $0.90^{100} = 0.0027\%$. Since at least one of the bits will almost certainly be random, simply XOR all the bits together and you'll have a random bit. Note that the specifications of the CK98 clock generator make it seem more likely than 10 percent that you'll see frequency deviations of 98 kHz.

If you actually use 200 intervals because of the need to set up before the next counting interval, the whole process would take six milliseconds per random bit produced.

205030-4152500T

14.51 Note (*computational efficiency of reduction modulo $b^t - c$*)

- (i) Suppose that x has $2t$ base b digits. If $l \leq t/2$, then Algorithm 14.47 executes step 2 at most $s = 3$ times, requiring 2 multiplications by c . In general, if l is approximately $(s-2)t/(s-1)$, then Algorithm 14.47 executes step 2 about s times. Thus, Algorithm 14.47 requires about sl single-precision multiplications.
- (ii) If c has few non-zero digits, then multiplication by c will be relatively inexpensive. If c is large but has few non-zero digits, the number of iterations of Algorithm 14.47 will be greater, but each iteration requires a very simple multiplication.

14.52 Note (*modifications*) Algorithm 14.47 can be modified if $m = b^t + c$ for some positive integer $c < b^t$: in step 2.2, replace $r \leftarrow r + r_i$ with $r \leftarrow r + (-1)^i r_i$.

14.53 Remark (*using moduli of a special form*) Selecting RSA moduli of the form $b^t \pm c$ for small values of c limits the choices of primes p and q . Care must also be exercised when selecting moduli of a special form, so that factoring is not made substantially easier; this is because numbers of this form are more susceptible to factoring by the special number field sieve (see §3.2.7). A similar statement can be made regarding the selection of primes of a special form for cryptographic schemes based on the discrete logarithm problem.

14.4 Greatest common divisor algorithms

Many situations in cryptography require the computation of the greatest common divisor (gcd) of two positive integers (see Definition 2.86). Algorithm 2.104 describes the classical Euclidean algorithm for this computation. For multiple-precision integers, Algorithm 2.104 requires a multiple-precision division at step 1.1 which is a relatively expensive operation. This section describes three methods for computing the gcd which are more efficient than the classical approach using multiple-precision numbers. The first is non-Euclidean and is referred to as the *binary gcd algorithm* (§14.4.1). Although it requires more steps than the classical algorithm, the binary gcd algorithm eliminates the computationally expensive division and replaces it with elementary shifts and additions. Lehmer's gcd algorithm (§14.4.2) is a variant of the classical algorithm more suited to multiple-precision computations. A binary version of the extended Euclidean algorithm is given in §14.4.3.

14.4.1 Binary gcd algorithm

14.54 Algorithm Binary gcd algorithm

INPUT: two positive integers x and y with $x \geq y$.

OUTPUT: $\gcd(x, y)$.

1. $g \leftarrow 1$.
 2. While both x and y are even do the following: $x \leftarrow x/2$, $y \leftarrow y/2$, $g \leftarrow 2g$.
 3. While $x \neq 0$ do the following:
 - 3.1 While x is even do: $x \leftarrow x/2$.
 - 3.2 While y is even do: $y \leftarrow y/2$.
 - 3.3 $t \leftarrow |x - y|/2$.
 - 3.4 If $x \geq y$ then $x \leftarrow t$; otherwise, $y \leftarrow t$.
 4. Return($g \cdot y$).
-

14.3.4 Reduction methods for moduli of special form

When the modulus has a special (customized) form, reduction techniques can be employed to allow more efficient computation. Suppose that the modulus m is a t -digit base b positive integer of the form $m = b^t - c$, where c is an l -digit base b positive integer (for some $l < t$). Algorithm 14.47 computes $x \bmod m$ for any positive integer x by using only shifts, additions, and single-precision multiplications of base b numbers.

14.47 Algorithm Reduction modulo $m = b^t - c$

INPUT: a base b , positive integer x , and a modulus $m = b^t - c$, where c is an l -digit base b integer for some $l < t$.

OUTPUT: $r = x \bmod m$.

1. $q_0 \leftarrow \lfloor x/b^t \rfloor$, $r_0 \leftarrow x - q_0 b^t$, $r \leftarrow r_0$, $i \leftarrow 0$.
2. While $q_i > 0$ do the following:
 - 2.1 $q_{i+1} \leftarrow \lfloor q_i c / b^t \rfloor$, $r_{i+1} \leftarrow q_i c - q_{i+1} b^t$.
 - 2.2 $i \leftarrow i + 1$, $r \leftarrow r + r_i$.
3. While $r \geq m$ do: $r \leftarrow r - m$.
4. Return(r).

14.48 Example (reduction modulo $b^t - c$) Let $b = 4$, $m = 935 = (32213)_4$, and $x = 31085 = (13211231)_4$. Since $m = 4^5 - (1121)_4$, take $c = (1121)_4$. Here $t = 5$ and $l = 4$. Table 14.9 displays the quotients and remainders produced by Algorithm 14.47. At the beginning of step 3, $r = (102031)_4$. Since $r > m$, step 3 computes $r - m = (3212)_4$. \square

i	$q_{i-1}c$	q_i	r_i	r
0	—	$(132)_4$	$(11231)_4$	$(11231)_4$
1	$(221232)_4$	$(2)_4$	$(21232)_4$	$(33123)_4$
2	$(2302)_4$	$(0)_4$	$(2302)_4$	$(102031)_4$

Table 14.9: Reduction modulo $m = b^t - c$ (see Example 14.48).

14.49 Fact (termination) For some integer $s \geq 0$, $q_s = 0$; hence, Algorithm 14.47 terminates.

Justification. $q_i c = q_{i+1} b^t + r_{i+1}$, $i \geq 0$. Since $c < b^t$, $q_i = (q_{i+1} b^t / c) + (r_{i+1} / c) > q_{i+1}$. Since the q_i 's are non-negative integers which strictly decrease as i increases, there is some integer $s \geq 0$ such that $q_s = 0$.

14.50 Fact (correctness) Algorithm 14.47 terminates with the correct residue modulo m .

Justification. Suppose that s is the smallest index i for which $q_i = 0$ (i.e., $q_s = 0$). Now, $x = q_0 b^t + r_0$ and $q_i c = q_{i+1} b^t + r_{i+1}$, $0 \leq i \leq s-1$. Adding these equations gives $x + \left(\sum_{i=0}^{s-1} q_i \right) c = \left(\sum_{i=0}^{s-1} q_{i+1} \right) b^t + \sum_{i=0}^s r_i$. Since $b^t \equiv c \pmod{m}$, it follows that $x \equiv \sum_{i=0}^s r_i \pmod{m}$. Hence, repeated subtraction of m from $r = \sum_{i=0}^s r_i$ gives the correct residue.

United States Patent & Trademark Office
Office of Initial Patent Examination

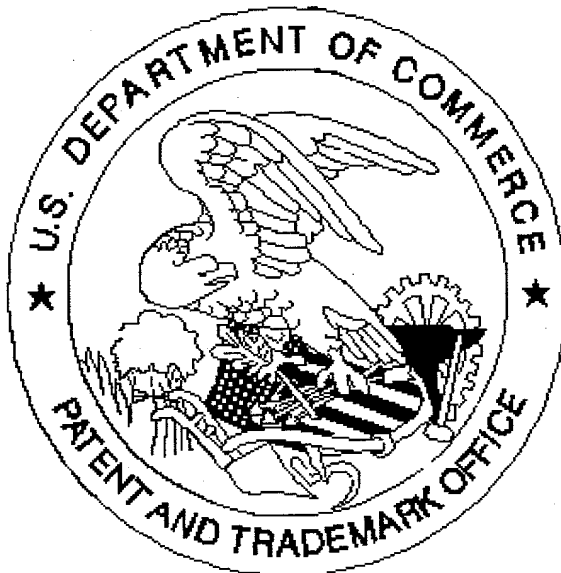
Application papers not suitable for publication

SN 10,092,943 Mail Date 3/5/02

- ☐ Non-English Specification
- ☐ Specification contains drawing(s) on page(s) _____ or table(s) _____
- ☐ Landscape orientation of text ☐ Specification ☐ Claims ☐ Abstract
- ☐ Handwritten ☐ Specification ☐ Claims ☐ Abstract
- ☐ More than one column ☐ Specification ☐ Claims ☐ Abstract
- ☐ Improper line spacing ☐ Specification ☐ Claims ☐ Abstract
- ☒ Claims not on separate page(s)
- ☐ Abstract not on separate page(s)
- ☐ Improper paper size -- Must be either A4 (21 cm x 29.7 cm) or 8-1/2"x 11"
- ☐ Specification page(s) _____ ☐ Abstract
- ☐ Drawing page(s) _____ ☐ Claim(s)
- ☐ Improper margins
- ☐ Specification page(s) _____ ☐ Abstract
- ☐ Drawing page(s) _____ ☐ Claim(s)
- ☐ Not reproducible
- | <u>Reason</u> | <u>Section</u> |
|---|--|
| <input type="checkbox"/> Paper too thin | <input type="checkbox"/> Specification page(s) _____ |
| <input type="checkbox"/> Glossy pages | <input type="checkbox"/> Drawing page(s) _____ |
| <input type="checkbox"/> Non-white background | <input type="checkbox"/> Abstract |
| | <input type="checkbox"/> Claim(s) |
- ☐ Drawing objection(s)
- ☐ Missing lead lines, drawing(s) _____
- ☐ Line quality is too light, drawing(s) _____
- ☐ More than 1 drawing and not numbered correctly
- ☐ Non-English text, drawing(s) _____
- ☐ Excessive text, drawing(s) _____
- ☐ Photographs capable of illustration, drawing(s) _____

205020-2462601

United States Patent & Trademark Office
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☒ Page(s) _____ of Abstract ^{was} ~~were~~ not present
for scanning. (Document title)

☐ Page(s) _____ of _____ were not present
for scanning. (Document title)

• Scanned copy is best available.

APPENDIX ~~SOME~~ pages are dark
and also have a dark line in some
pages

2050ED E462600T